BRAZILIAN STANDARD

ABNT NBR 15606-2

Second edition
2011.05.27

Valid from
2011.06.27

Digital terrestrial television – Data coding and transmission specification for digital broadcasting
Part 2: Ginga-NCL for fixed and mobile receivers – XML application language for application coding

ICS 33.160.01

ISBN 978-85-07-02878-9

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS

Reference number
ABNT NBR 15606-2:2011
288 pages

© ABNT 2011

© ABNT 2011
All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ABNT.

ABNT office
Av.Treze de Maio, 13 - 28º andar
20031-901 - Rio de Janeiro - RJ
Tel.: + 55 21 3974-2300
Fax: + 55 21 2220-1762
abnt@abnt.org.br
www.abnt.org.br

Published in Brazil

© ABNT 2011 - All rights reserved

Contents

Pages

© ABNT 2011 - All rights reserved

© ABNT 2011 - All rights reserved

© ABNT 2011 - All rights reserved

© ABNT 2011 - All rights reserved

Foreword

Associação Brasileira de Normas Técnicas (ABNT) is the Brazilian Standardization Forum. Brazilian Standards, which content is responsability of the Brazilian Committees (Comitês Brasileiros – ABNT/CB), Sectorial Standardization Bodies (Organismos de Normalização Setorial – ABNT/ONS) and Special Studies Committees (Comissões de Estudo Especiais – ABNT/CEE), are prepared by Study Committees (Comissões de Estudo – CE), made up of representants from the sectors involved including: producers, consumers and neutral entities (universities, laboratories and others).

Brazilian Standards are drafted in accordance with the rules given in the ABNT Directives (Diretivas), Part 2.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ABNT shall not be held responsible for identifying any or all such patent rights.

ABNT NBR 15606-2 was prepared within the purview of the Special Studies Committees of Digital Television (ABNT/CEE-00:001.85). The Draft Standard was circulated for National Consultation in accordance with ABNT Notice (Edital) nº 09, from September 06, 2007 to November 05, 2007, with the number Draft 00:001.85-006/2. Its Amendment Draft was circulated for National Consultation in accordance with ABNT Notice (Edital) nº 01, from January 13th, 2011, to March 14th, 2011, with the number Amendment Draft ABNT NBR 15606-2.

Should any doubts arise regarding the interpretation of the English version, the provisions in the original text in Portuguese shall prevail at all time.

This standard is based on the work of the Brazilian Digital Television Forum as established by the Presidential Decree number 5.820 of June, 29th 2006.

ABNT NBR 15606 consists of the following parts, under the general title "*Digital terrestrial television — Data coding and transmission specifications for digital broadcasting*":

— Part 1: Data coding specification;

— Part 2: Ginga-NCL for fixed and mobile receivers – XML application language for application coding;

— Part 3: Data transmission specification;

— Part 4: Ginga-J – The environment for the execution of imperative applications;

— Part 5: Ginga-NCL for portable receivers – XML application language for application coding

— Part 6: Java DTV 1.3;

— Part 7: Ginga-NCL: Operational guidelines to ABNT NBR 15606-2 and ABNT NBR 15606-5.

This second edition incorporates the Amendment 1 from 2011.05.27, and cancels and replaces the previous edition (ABNT NBR 15606-2:2007).

This Standard is the English version of the ABNT NBR 15607-1:2011.

This version in English was published in 2011.07.11.

© ABNT 2011 - All rights reserved

Introduction

The Brazilian Standardization Forum (ABNT – Associação Brasileira de Normas Técnicas) draws attention to the fact that it is claimed that compliance with this document may involve the use of the property right concerning the NCL mentioned in 5.1.

ABNT takes no position concerning the evidence, validity and scope of this property right.

The holder of this property right has assured to ABNT that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ABNT. Information may be obtained from:

Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Transferência de Tecnologia

Rua Marquês de São Vicente, 225 – Gávea, 22451-900 - Rio de Janeiro - RJ – Brasil

ABNT draws attention to the possibility that some of the elements of this document may be the subject of patent rights, other than those identified above. ABNT shall not be held responsible for identifying any or all such patent rights.

This Standard provides an XML application language that allows authors to write interactive multimedia presentations. This component of ABNT NBR 15606 is part of the data coding specifications of the Brazilian digital television system (SBTVD) and comprises the language specification used by the presentation engine Ginga-NCL of the SBTVD middleware, named Ginga.

Using this language, called NCL (Nested Context Language), an author may describe the temporal behavior of a multimedia presentation, associate hyperlinks (user interaction) with media objects, define alternatives for presentation (adaptation), and describe the layout of the presentation on multiple devices.

This Standard is firstly intended to be used by entities writing terminal specifications and/or standards/recommendations based on Ginga. Secondly, it is intended for developers of applications that use the Ginga functionalities and API. Ginga aims to ensure interoperability of Ginga applications and running on different platforms supporting it.

Ginga applications are classified into two categories depending upon whether the initial application content processed is of a declarative or an imperative nature. These categories of applications are referred to as declarative and imperative applications, respectively. Application environments are similarly classified into two categories depending upon whether they process declarative or imperative applications, and are called Ginga-NCL and Ginga-J, respectively.

It is important to note that, for fixed and mobile receivers, to a Ginga-NCL-only or Ginga-J-only implementation is prohibited the claim any kind of SBTVD conformance. This ensures that Ginga always offer backward-compatible profiles.

This Standard does not specify the implementation of application environments in a compliant receiver. A receiver manufacturer may implement both environments as a single subsystem; alternatively, both environments may be implemented as distinct subsystems with well-defined, internal inter-environment interfaces.

© ABNT 2011 - All rights reserved

Digital terrestrial television – Data coding and transmission specification for digital broadcasting – Part 2: Ginga — NCL for fixed and mobile receivers — XML application language for application coding

1 Scope

This part of ABNT NBR 15606 specifies an XML application language, named NCL (Nested Context Language), the declarative language of the middleware Ginga, and the data coding and transmission for digital broadcasting.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ABNT NBR 15601, *Digtial terrestrial television – Transmission system*

ABNT NBR 15603-2:2007*, Digital terrestrial television – Multiplexing and service information (SI) – Part 2: Data structure and definitions of basic information of SI*

*AB*NT NBR 15606-1, *Digital terrestrial television – Data coding and transmission specification for digital broadcasting – Part 1: Data coding specification*

ABNT NBR 15606-3, *Digital terrestrial television – Data coding and transmission specification for digital broadcasting – Part 3: Data transmission specification*

ISO 639-1, *Codes for the representation of names of languages - Part 1: Alpha-2 code*

ISO 8859-1, *Information technology – 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet N° 1*

ISO/IEC 11172-1*, Coding of moving pictures and associated audio for digital storage mediaat up to about 1,5 Mbit/s – Part 1: Systems*

ISO/IEC 11172-2*, Coding of moving pictures and associated audio for digital storage mediaat up to about 1,5 Mbit/s – Part 2: Video*

ISO/IEC 11172-3*, Coding of moving pictures and associated audio for digital storage mediaat up to about 1,5 Mbit/s – Part 3: Audio*

ISO/IEC 13818-1, *Information technology – Generic coding of moving pictures and associated audio information – Part 1: Systems*

ISO/IEC 13818-2, *Information technology – Generic coding of moving pictures and associated audio information – Part 2: Video*

ISO/IEC 13818-3, *Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio*

ISO/IEC 13818-6, *Information technology – Generic coding of moving pictures and associated audio information – Part 6: Extensions for DSM-CC*

ISO/IEC 13818-7, *Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC)*

© ABNT 2011 - All rights reserved

ISO/IEC 14496-3, *Information technology – Coding of audio-visual objects – Part 3: Audio*

ECMA 262*, ECMAScript language specification*

3 Terms and definitions

For the purpose of this part of ABNT NBR 15606, the following terms and definitions apply.

3.1
application environment
context or software environment in which an application is processed

3.2
declarative application environment
environment that supports the processing of declarative applications

NOTE An NCL formatter (user agent) is an example of a declarative application environment.

3.3
imperative application environment
environment that supports the processing of imperative applications

3.4
DOM API
API that defines the logical structure of an XML document and the way to access or manipulate an XML document

NOTE This API is an interface independent of platforms and languages, and follows the DOM Model (Document Object Model).

3.5
application
information that expresses a specific set of observable behaviors

3.6
declarative application
application which is started by and primarily makes use of declarative information to express its behavior

NOTE An NCL document instance is an example of a declarative application.

3.7
hybrid application
hybrid declarative application or a hybrid imperative application

3.8
hybrid declarative application
declarative application that makes use of active object content

NOTE An NCL document with an embedded Java Xlet is an example of a hybrid declarative application.

3.9
hybrid imperative application
imperative application with declarative content

NOTE A Java Xlet that creates and causes the display of an NCL document instance is an example of a hybrid imperative application.

© ABNT 2011 - All rights reserved

3.10
native application
an intrinsic function implemented by a receiver platform

NOTE A closed captioning display is an example of a native application.

3.11
imperative application
application that is started by and primarily makes use of imperative information to express its behaviour

NOTE A Java program is an example of a imperative application.

3.12
persistent storage
memory available that may be read or written to by an application and may outlive the application's own life

NOTE Persistent storage can be volatile or non-volatile.

3.13
attribute
parameter that represents the character of a property

3.14
attribute of an element
property of an XML element

3.15
main audio

basic audio stream whose component_tag is equal to 0x010 for full-seg receiver and 0x85 for one-seg receiver

3.16
author
person who writes NCL documents

3.17
interactive channel
return channel
communication mechanism which provides connection between the receiver and a remote server

3.18
character
specific "letter" or other identifiable symbol

EXAMPLE "A"

3.19
data carousel
method that sends out any set of data cyclically; the data may thus be downloaded via broadcasting in a time interval as long as needed

[ISO/IEC 13818-6:2001]

3.20
application life-cycle
time period from the moment an application is loaded until the moment it is destroyed

© ABNT 2011 - All rights reserved

3.21
character encoding
mapping between an integer input value and the textual character that is represented by this mapping

3.22
active object content
type of content which takes the form of an executable program

NOTE A compiled Java Xlet is an example of an active object content.

3.23
NCL content
set of information that consists of an NCL document and a group of data including objects (media or execution objects) accompanying the NCL document

3.24
digital storage media command and control
DSM-CC
control method, which provides access to a file or a stream in digital interactive services

[ISO/IEC 13818-6:2001]

3.25
document type definition
DTD
declaration that describes a family of XML documents

3.26
ECMAScript
programming language defined in the ECMA 262

3.27
element
document structuring unit delimited by tags

NOTE An element is usually delimited by a start tag and an end tag, except for an empty element that is delimited by an empty element tag

3.28
property element
NCL element that defines a property name and its associated value

3.29
application entity
unit of information that expresses some portion of an application

3.30
event
occurrence in time that may be instantaneous or have a measurable duration

3.31
media player
identifiable component of an application environment which decodes or executes a specific content type

3.32
eXtensible HTML
XHTML
extended version of HTML

NOTE In the XHTML specification, an HTML document is recognized as an XML application.

© ABNT 2011 - All rights reserved

3.33
authoring tool
tool to help authors to generate NCL documents

3.34
font
mechanism that allows the specific rendering of a particular character

EXAMPLE Tiresias, 12 points.

NOTE In practice, a font format incorporates some aspects of a character encoding.

3.35
NCL formatter
software component that is in charge of receiving the specification of an NCL document and controlling its presentation, trying to guarantee that author-specified relationships among media objects are respected.

NOTE Document renderer, user agent and player are other names used with the same meaning of document formatter.

3.36
transport stream
refers to the MPEG-2 transport stream syntax for the packetization and multiplexing of video, audio and data signals for digital broadcast systems

3.37
elementary stream
ES
a basic stream that contains video data, audio data, or private data.

NOTE A single elementary stream is carried in a sequence of PES packets with one and only one stream_id.

3.38
application manager
entity that is responsible for managing the lifecycle of applications and that manages applications running in both the presentation engine and the execution engine

3.39
packet identifier
PID
unique integer value used to associate elementary streams of a program in a single or multi-program transport stream

3.40
service information
SI
data which describes programs and services

3.41
program specific information
PSI
normative data which is necessary for the demultiplexing of transport streams and the successful regeneration of programs

[ISO/IEC 13818-1:2005]

© ABNT 2011 - All rights reserved

3.42
application programming interface
API
consists of software libraries that provide uniform access to system services

3.43
markup language
formalism that describes a class of documents which employ markup in order to delineate the document's structure, appearance or other aspects

3.44
scripting language
language used to describe an active object content which is embedded in NCL documents and in HTML documents

3.45
locator
identifier that provides a reference to an application or resource

3.46
presentation engine
subsystem in a receiver that evaluates and presents declarative applications, consisting of media contents, such as audio, video, graphics and text, based on presentation rules defined in the presentation engine

NOTE A presentation engine is responsible for controlling the presentation behavior and initiating other processes in response to user input and other events.

EXAMPLE HTML browser and NCL formatter.

3.47
execution engine
subsystem in a receiver that evaluates and executes imperative applications consisting of computer language instructions and associated data and media content

NOTE An execution engine can be implemented grouping an operating system, language compilers, interpreters, and Application Programming Interfaces (APIs), which a imperative application may use to present audiovisual content, interact with a user, or execute other tasks that are not evident to the user.

EXAMPLE The JavaTV software environment, using the Java programming language and byte code interpreter, JavaTV API and a Java Virtual Machine for program execution.

3.48
method
a function that is associated with an object and is allowed to manipulate the object's data

3.49
NCL node
refers to a <media>, <context>, <body>, or <switch> element of NCL

3.50
normal play time
NPT
absolute temporal coordinate which represents the position in a stream

3.51
media object
collection of named pieces of data that may represent a media content or a program written in a specific language

3.52
visual media object
any NCL media object, represented by a <media> element, whose content produces a visual presentation, when the object is started

© ABNT 2011 - All rights reserved

3.53
profile
specification for a class of capabilities providing different levels of functionality in a receiver

3.54
one-seg profile
characterizes the service that may be received by a narrow-band tunner (430 KHz), therefore, with power saving

NOTE The one-seg profile is also known as portable profile.

3.55
full-seg profile
characterizes the service that needs a broad-band demodulator (5,7 MHz) to be received

NOTE Depending on the transmission configuration and on specific receiver functionalities, the service may be received by mobile receivers or only by fixed receivers, however, without the benefits of power saving. The video resolution may be high definition or standard definition.

3.56
plug-in
set of functionality which may be added to a generic platform in order to provide additional functionality

3.57
receiver platform
platform
the receiver's hardware, operating system and native software libraries of the manufacturer's choice

3.58
resource
network data object or a service which is uniquely identified in a network

3.59
local file system
file system provided by the local receiver platform

3.60
lifetime of an application
time from which the application is loaded to the time the application is destroyed

3.61
uniform resource identifier
URI
addressing method to allow access to objects in a network

3.62
user agent
any program that interprets a document written in NCL language

NOTE A user agent may display a document, trying to guarantee that author-specified relationships among media objects are respected, read it aloud, cause it to be printed, convert it to another format etc.

3.63
user
person who interacts with a user agent to view, hear, or otherwise use an NCL document

3.64
end user
individual operating or interacting with a receiver

© ABNT 2011 - All rights reserved

4 Abbreviations

For the purposes of this part of ABNT NBR 15606, the following abbreviations apply.

API	Application Programming Interface
BML	Broadcast Markup Language
CLUT	Color Look-up Table
CSS	Cascading Style Sheets
DOM	Document Object Model
DSM-CC	Digital Storage Media Command and Control
DTD	Document Type Definition
DTV	Digital Television
DVB	Digital Video Broadcasting
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JPEG	Joint Photographic Expert Group
MIME	Multipurpose Internet Mail Extensions
MNG	Multiple Network Graphics
MPEG	Moving Picture Expert Group
NCL	Nested Context Language
NCM	Nested Context Model
NIT	Network Information Table
NPT	Normal Play Time
OS	Operating System
PAT	Program Association Table
PES	Packetized Elementary Stream
PID	Packet Identifier
PMT	Program Map Table
PNG	Portable Network Graphics
PSI	Program Specific Information
SBTVD	Brazilian digital television system
SMIL	Synchronized Multimedia Integration Language
TS	Transport Stream
UCS	Universal (Coded) Character Set
URI	Universal Resource Identifier
URL	Universal Resource Locator
UTF	UCS Transformation Coding
XHTML	eXtensible HTML
XML	Extensible Markup Language
W3C	World-Wide Web Consortium

© ABNT 2011 - All rights reserved

5 Ginga architecture

5.1 Ginga main modules

The universe of Ginga applications can be partitioned into a set of declarative applications and a set of imperative applications. A declarative application is an application whose initial entity is of a declarative content type. A imperative application is an application whose initial entity is of a imperative content type. A purely declarative application is one whose every entity is of a declarative content type. A purely imperative application is one whose every entity is of a imperative content type. A hybrid application is one whose entity set contains entities of both declarative and imperative content types. A Ginga application needs not be purely declarative nor imperative.

In particular, declarative applications often make use of script content, which is imperative in nature. Furthermore, a declarative application may refer to an embedded Java TV Xlet. Similarly, a imperative application may refer to a declarative content, such as graphic content, or may construct and initiate the presentation of declarative content. Therefore, either type of Ginga application may make use of facilities of both declarative and imperative application environments.

Ginga-NCL is a logical subsystem of the Ginga system that processes NCL documents [1)] . A key component of Ginga-NCL is the declarative content decoding engine (NCL formatter). Other important modules are the XHTML user agent, which includes a stylesheet (CSS) and ECMAScript interpreter, and the Lua engine, which is responsible for interpreting Lua scripts (see Annex B).

Ginga-J is a logical subsystem of the Ginga system that processes active object content. A key component of the imperative application environment is the imperative content execution engine, composed of a Java virtual machine.

Common content decoders serve both imperative and declarative application needs for the decoding and presentation of common content types such as PNG, JPEG, MPEG and other formats. The Ginga common core is composed of common content decoders and procedures to obtain contents transported in MPEG-2 transport dtreams and via the interactive channel. The Ginga common core shall also support the conceptual display model as described in ABNT NBR 15606-1.

The architecture (see Figure 1) and facilities of the Ginga Recommendation are intended to apply to transmission systems and receivers for terrestrial broadcast. In addition, the same architecture and facilities can be applied to other transport systems (such as satellite or cable television systems).

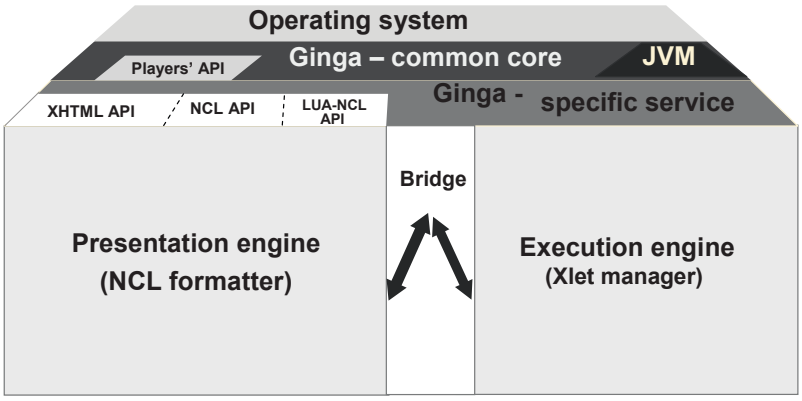

Figure 1 – Ginga architecture

1) NCL is a trade and its specification is an intellectual property of PUC-Rio (INPI Technology Transfer Department - Nº 0007162-5; 20/12/2005).

© ABNT 2011 - All rights reserved

5.2 Interaction with the native environment

In general, Ginga is independent of any native applications that may also choose to use the graphics plane. These include but are not limited to: closed captioning, CAS messages, receiver menus and native program guides.

Native applications may take precedence over Ginga applications. Closed captioning and emergency messaging shall take precedence over the Ginga system.

Some native applications, such as closed captioning, present a special case where the native application can be active for long periods concurrently with Ginga applications.

6 Interoperability with other digital television system declarative environments – XHTML objects embedded in NCL presentations

6.1 NCL as glue language

All presentation engines of the three main digital televison systems use an XHTML-based language.

XHTML is a media-based declarative language, which means that the structure is defined by the relationships among XHTML objects (XHTML documents or objects enclosed in XHTML documents) that are embedded in the document's media content. XHTML may thus be classified as a markup language: a formalism that describes a class of documents which employ markup in order to delineate the document's structure, appearance and other aspects.

Reference relationships defined by XHTML links are the focus of the XHTML declarative language. Other relationship types, like spatio-temporal synchronization relationships and alternative relationships (media adaptability), are usually defined using imperative languages (for example, ECMAScript).

Unlike HTML or XHTML, NCL has a stricter separation between a document's (or application's) content and structure, providing non-invasive control of presentation linking and layout.

The focus of the NCL declarative language is broader than the XHTML counterpart. Generalized spatio-temporal synchronization, defined by NCL links; adaptability, defined by NCL switch and descriptor switch elements; and support for multiple exhibition devices, defined by NCL regions, is the focus of the NCL declarative language. User interaction is treated just as a special case of temporal synchronization.

As NCL has a stricter separation between content and structure, NCL does not define any media itself. Instead, it defines the glue that holds media together in multimedia presentations.

An NCL document 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. In this sense, it may have image objects (GIF, JPEG etc.), video objects (MPEG, MOV etc.), audio objects (MP3, WMA etc.), text objects (TXT, PDF etc.), execution objects (Xlet, Lua etc.) among others, as NCL media objects. Which media objects are supported depends on the media players that are coupled in the NCL formatter (NCL player). One of these players is the MPEG-4 decoder/player, usually implemented in hardware in the digital television receiver. In this way, the main MPEG-4 video and audio are treated like all other media objects that may be related using NCL.

Another NCL media object that shall be supported is the XHTML-based media object. NCL does not substitute but embed XHTML-based documents (or objects). As with other media objects, which XHTML-based language has support in an NCL formatter is an implementation choice, and therefore it depends on which XHTML browser acts as a media player coupled in the NCL formatter.

As a consequence, it is possible to have BML browsers, DVB-HTML browsers and ACAP-X browsers embedded in an NCL document player. It is even possible to have all of them. It is also possible to receive a browser code through datacasting and install it as a plug-in (usually a Java plug-in).

© ABNT 2011 - All rights reserved

It is also possible to have a harmonization browser implemented, and receiving the complementary part, if needed, as a plug-in, in order to convert the XHTML player into one of the several DTV browser standards.

In the extreme case, an NCL document may be reduced to having only one XHTML media object. In this case, the NCL document player acts nearly like an XHTML browser. That is, just like any browser of the other aforementioned standards.

No matter the case, the XHTML-based browser implementation shall be a consequence of the following requirements:

— interoperability;

— robustness;

— alignment with W3C specifications;

— rejection of non-conformant content;

— compatibility with the Ginga security model;

— minimization of the redundancy with existing Ginga-J technology;

— minimization of the redundancy with existing NCL facilities;

— precise content layout control mechanisms;

— support of different pixel aspect ratios.

In order to support all XHTML-based browser facilities defined by other DTV standards, all SBTVD specifications related to datacasting should also support the facilities defined for those browsers, such as the transport of stream events, for example.

Although an XHTML-based browser shall be supported, the use of XHTML elements to define relationships (including XHTML links) should be dissuaded when authoring NCL documents. Structure-based authoring should be emphasized for the well-known reasons largely reported in the literature.

During the exhibition of media-object contents, several events are generated (see 7.2.8). Some examples of events are the presentation of marked segments of media-object content, the selection of a marked content segment etc. Events may generate actions on other media objects, like to start or stop their presentations. Hence, events shall be reported by media players to the NCL formatter that, in its turn, can generate actions to be applied to these or other players. Ginga-NCL defines an adapter API (see Clause 8) to standardize the interface between the Ginga-NCL formatter and each specific player.

When any media player, in particular an XHTML-based browser, is integrated to the Ginga-NCL formatter, it shall support the adapter API. Therefore, for some media players, including XHTML-based browsers, an adapter module can be necessary to accomplish the integration.

For live editing, Ginga-NCL has also defined NCL stream events in order to support live generated events in stream media, in particular the main program stream video. These events are a generalization of the same concept found in other standards, like for example the *bevents* of BML. Although an XHTML-based browser shall be supported, the use of XHTML elements to define relationships (including stream events) should be dissuaded in authoring NCL documents, for the same motivation, that is, structure-based authoring should be emphasized for the well-known reasons largely reported in the literature.

© ABNT 2011 - All rights reserved

6.2 XHTML-based content format

Common content formats shall be adopted for production and exchange of multimedia content, as defined in ABNT NBR 15606-1. In addition, specification of harmonized XHTML-based content formats in the declarative application environment is also required for interactive television applications.

NOTE This Standard follows the ITU-T Recommendation J.201 in order to identify the functional commonality among the declarative application environments for interactive television applications specified by DVB-HTML, ACAP-X and BML.

Common elements and API at the syntactic level of the XHTML-based media-objects embedded in NCL applications should be specified in order to assist authors to create XHTML-based content.

Any XHTML-based media object implementation in conformance with this Standard shall at least support all common XML markups and stylesheet properties for the BML for basic services ("fixed terminal profile"), ACAP-X and DVB-HTML, as defined in the next section. Common features of ECMAScript native objects and DOM API, for the BML for basic services ("fixed terminal profile"), ACAP-X and DVB-HTML, should also be supported.

6.3 Harmonization of XHTML-based content format

6.3.1 XML markups

NOTE XHTML-based media-objects of NCL conform to the “Modularization of XHTML” W3C Recommendation and their common XML markups are defined in the ITU Recommendation J.201.

The common XML markup modules may be:

— structure;

— text;

— hypertext;

— list;

— presentation;

— bidirectional text;

— forms;

— image;

— client-side image map;

— object;

— frames;

— target;

— meta information;

— scripting;

— stylesheet;

— style attribute;

— link;

— base.

© ABNT 2011 - All rights reserved

XHTML attribute collections are defined as presented in Table 1. The common XML markups for the BML for basic services ("fixed terminal profile"), ACAP-X and DVB-HTML, which shall be supported by any implementation, are listed in Table 2, together with Ginga extensions.

Table 1 – Attribute collections

Collection name	Attributes in collection	Attribute condition
Core	class (NMTOKENS)	Required
	id (ID)	Required
	title (CDATA)	–
I18N	xml:lang (CDATA)	Required
Events	onclick (Script)	Required
	ondblclick (Script)	–
	onmousedown (Script)	–
	onmouseup (Script)	–
	onmouseover (Script)	–
	onmousemove (Script)	–
	onmouseout (Script)	–
	onkeypress (Script)	–
	onkeydown (Script)	Required
	onkeyup (Script)	Required
Style	style (CDATA)	Required
Common	Core + Events + I18N + Style	

© ABNT 2011 - All rights reserved

Table 2 – Common XML markup elements

Module		Element	Element condition	Attribute	Atribute condition
Core	Structure	body	Required	%Common.attrib	
				%Core.attrib	Required
				%I18n.attrib	Required
				%Events.attrib	–
		head	Required	%I18n.attrib	Required
				profile	–
		html	Required		
		title	Required	%I18n.attrib	Required
	Text	abbr	–		
		acronym	–		
		address	–		
		blockquote	–		
		br	Required	%Core.attrib	Required
		cite	–		
		code	–		
		dfn	–		
		div	Required	%Common.attrib	Required
		em	–		
		h1	Required	%Core.attrib	Required
		h2	Required	%Core.attrib	Required
		h3	Required	%Core.attrib	Required
		h4	Required	%Core.attrib	Required
		h5	Required	%Core.attrib	Required
		h6	Required	%Core.attrib	Required
		kbd	–		
		p	Required	%Common.attrib	Required
		pre	–		
		q	–		
		samp	–		
		span	Required	%Common.attrib	Required
		strong	–		
		var	–		
	Hypertext	a	Required	%Common.attrib	Required
				accesskey	Required
				charset	Required
				href	Required
				hreflang	–
				rel	–
				rev	–
				tabindex	–
				type	–
	List	dl	–		
		dt	–		
		dd	–		
		ol	–		
		ul	–		
		li	–		

© ABNT 2011 - All rights reserved

Table 2 (continuation)

Module		Element	Element condition	Attribute	Atribute condition
Applet		applet	–		
		param	–		
Text extension	Presentation	b	–		
		big	–		
		hr	–		
		i	–		
		small	–		
		sub	–		
		sup	–		
		tt	–		
	Edit	del	–		
		ins	–		
	Bi-directional text	bdo	–		
Forms	Basic forms	form	–		
		input	–		
		label	–		
		select	–		
		option	–		
		textarea	–		
	Forms	form	Required	%Common.attrib	Required
				action	Required
				method	Required
				enctype	Required
				accept-charset	Required
				accept	Required
				name	Required
		input	Required	%Common.attrib	Required
				accesskey	Required
				checked	–
				disabled	Required
				readonly	Required
				maxlength	Required
				alt	–
				name	–
				size	Required
				src	–
				tabindex	–
				accept	–
				type	Required
				value	Required
		select	–		
		option	–		
		textarea	–		
		button	–		
		fieldset	–		
		label	–		
		legend	–		
		optgroup	–		

© ABNT 2011 - All rights reserved

Table 2 (continuation)

Module		Element	Element condition	Attribute	Atribute condition
Table	Basic tables	caption	–		
		table	–		
		td	–		
		th	–		
		tr	–		
	Tables	caption	–		
		table	–		
		td	–		
		th	–		
		tr	–		
		col	–		
		colgroup	–		
		tbody	–		
		thead	–		
		tfoot	–		
Image		img	–		
Client side map		a&	–		
		area	–		
		img&	–		
		input&	–		
		map	–		
		object&	–		
Server side image map		img&	–		
		input&	–		
Object		object	Required	%Common.attrib	Required
				archive	–
				classid	–
				codebase	–
				codetype	–
				data	Required
				declare	–
				height	Required
				name	–
				standby	–
				tabindex	–
				type	Required
				width	Required
		param	–		
Frames		frameset	–		
		frame	–		
		noframe	–		
Target		a&	–		
		area&	–		
		base&	–		
		link&	–		
		form&	–		
IFrame		iframe	–		

© ABNT 2011 - All rights reserved

Table 2 (continuation)

Module	Element	Element condition	Attribute	Atribute condition
Intrinsic events	a&	Required		
	area&	–		
	frameset&	–		
	form&	–		
	body&	–		
	label&	–		
	input&	–		
	select&	–		
	textarea&	–		
	button&	–		
Metainformation	meta	Required	%I18n.attrib	–
			http-equiv	–
			name	Required
			content	Required
			scheme	–
Scripting	noscript			
	script	Required	charset	Required
			type	Required
			src	–
			defer	–
Stylesheet	style	Required	%I18n.attrib	Required
			id	–
			type	Required
			media	Required
			title	–
Style attribute		Required		
Link	link	Required		
Base	base	–		

6.3.2 Stylesheet

The common stylesheet (CSS) properties are listed in Table 3.

© ABNT 2011 - All rights reserved

Table 3 – Common stylesheet properties

background	clear	outline-color
background-attachment	clip	outline-style
background-color	color	outline-width
background-image	content	overflow
background-position	counter-increment	padding
background-repeat	counter-reset	padding-bottom
border	display	padding-left
border-bottom	float	padding-right
border-bottom-color	font	padding-top
border-bottom-style	font-family	position
border-bottom-width	font-size	right
border-color	font-style	text-align
border-left	font-variant	text-decoration
border-left-color	font-weight	text-indent
border-left-style	height	text-transform
border-left-width	left	top
border-right	letter-spacing	vertical-align
border-right-color	line-height	visibility
border-right-style	list-style	white-space
border-right-width	list-style-image	width
border-style	list-style-position	word-spacing
border-top	list-style-type	z-index
border-top-color	margin	nav-down
border-top-style	margin-bottom	nav-index
border-top-width	margin-left	nav-left
border-width	margin-right	nav-right
bottom	margin-top	nav-up
caption-side	outline	----

The common stylesheet properties for BML for basic services, ACAP-X and DVB-HTML, which shall be supported by any implementation, are listed in Table 4.

© ABNT 2011 - All rights reserved

Table 4 – Common stylesheet CSS 2 properties

Property	Property condition
Value assignment/Inheritance	
@import	–
!important	–
Media type	
@media	Required
box model	
margin-top	–
margin-right	–
margin-bottom	–
margin-left	–
margin	Required
padding-top	Required
padding-right	Required
padding-bottom	Required
padding-left	Required
padding	Required
border-top-width	–
border-right-width	–
border-bottom-width	–
border-left-width	–
border-width	Required
border-top-color	–
border-right-color	–
border-bottom-color	–
border-left-color	–
border-color	Required
border-top-style	–
border-right-style	–
border-bottom-style	–
border-left-style	–
border-style	Required
border-top	–
border-right	–
border-bottom	–
border-left	–
border	Required
Visual formatting model	
position	Required
left	Required
top	Required
width	Required
height	Required
z-index	Required
line-height	Required
vertical-align	–
display	Required
bottom	–
right	–
float	–
clear	–
direction	–

© ABNT 2011 - All rights reserved

Table 4 (continuation)

Property	Property condition
unicode-bidi	–
min-width	–
max-width	–
min-height	–
max-height	–
Other visual effects	
visibility	Required
overflow	Required
clip	–
Generated content/Auto numbering/List	
content	–
quotes	–
counter-reset	–
counter-increment	–
marker-offset	–
list-style-type	–
list-style-image	–
list-style-position	–
list-style	–
Page media	
"@page"	–
size	–
marks	–
page-break-before	–
page-break-after	–
page-break-inside	–
page	–
orphans	–
widows	–
Background	
background	–
background-color	–
background-image	Required
background-repeat	Required
background-position	–
background-attachment	–
Font	
color	Required
font-family	Required
font-style	Required
font-size	Required
font-variant	Required
font-weight	Required
font	Required
font-stretch	–
font-adjust	–
Text	
text-indent	–
text-align	Required
text-decoration	–

© ABNT 2011 - All rights reserved

Table 4 (continuation)

Property	Property condition
text-shadow	–
letter-spacing	Required
word-spacing	–
text-transform	–
white-space	Required
Pseudo class/ Pseudo element	
:link	–
:visited	–
:active	Required
:hover	–
:focus	Required
:lang	–
:first-child	–
:first-line	–
:first-letter	–
:before	–
:after	–
Table	
caption-side	–
border-collapse	–
border-spacing	–
table-layout	–
empty-cells	–
speak-header	–
User interface	
outline-color	–
outline-width	–
outline-style	–
outline	–
cursor	–
voice style sheet	
volume	–
speak	–
pause-before	–
pause-after	–
pause	–
cue-before	–
cue-after	–
cue	–
play-during	–
azimuth	–
elevation	–
speech-rate	–
voice-family	–
pitch	–
pitch-range	–
stress	–
richness	–
speak-punctuation	–
peak-numeral	–

© ABNT 2011 - All rights reserved

Table 4 (continuation)

Property	Property condition
extended property	
clut	–
color-index	–
background-color-index	–
border-color-index	–
border-top-color-index	–
border-right-color-index	–
border-bottom-color-index	–
border-left-color-index	–
outline-color-index	–
resolution	–
display-aspect-ratio	–
grayscale-color-index	–
nav-index	–
nav-up	–
nav-down	–
nav-left	–
nav-right	–
used-key-list	–

The following restrictions shall be applied to display property:

— only block elements may be applied for <p>, , <body>, <input>, and <object>;

— only inline values may be applied for
, <a>, and .

Moreover, the following restrictions shall be applied to position property:

— only absolute values may be applied for <p>, , <input> and <object>;

— only static values may be applied for
, , and <a>.

The common CSS selectors for BML for basic services, ACAP-X and DVB-HTML, which shall be supported by any implementation, are the following:

— universal;

— type;

— class;

— id;

— dynamic (:active and :focus).

6.3.3 ECMAScript

When implemented, the ECMAScript engine should support the common native objects for BML for basic services, ACAP-X and DVB-HTML, listed in Table 5. As a restriction, number type supports integer operations only.

© ABNT 2011 - All rights reserved

Table 5 – Common native objects

Object	Method, properties	Operation condition
(global)		
	NaN	Required
	Infinity	–
	eval(x)	–
	parseInt(string, radix)	Required
	parseFloat(string)	–
	escape(string)	–
	unescape(string)	–
	isNaN(number) O	Required
	isFinite(number)	–
Object		All required
	prototype	Required
	Object([value])	Required
	new Object([value])	Required
Object.prototype		All required
	constructor	Required
	toString()	Required
	valueOf()	Required
Function		
	prototype	Required
	Length	Required
	Function(p1, p2, . . . , pn, body)	–
	new Function(p1, p2, . . . , pn, body)	–
Function.prototype		All required
	constructor	Required
	toString()	Required
Array		All required
	prototype	Required
	Length	Required
	Array(item0, item1, . . .)	Required
	new Array(item0, item1, . . .)	Required
	new Array([len])	Required
Array.prototype		All required
	constructor	Required
	toString()	Required
	join([separator])	Required
	reverse()	Required
	sort([comparefn])	Required
	constructor	Required
String		All required
	prototype	Required
	Length	Required
	String([value])	Required
	new String([value])	Required
	String.fromCharCode(char0[, char1, . . .])	Required

© ABNT 2011 - All rights reserved

Table 5 (continuation)

Object	Method, properties	Operation condition
String.prototype		All required
	constructor	Required
	toString()	Required
	valueOf()	Required
	charAt(pos)	Required
	charCodeAt(pos)	Required
	indexOf(searchString, position)	Required
	lastIndexOf(searchString, position)	Required
	split(separator)	Required
	substring(start [,end])	Required
	toLowerCase()	Required
	toUpperCase()	Required
Boolean		All required
	prototype	Required
	Boolean([value])	Required
	new Boolean([value])	Required
Boolean.prototype		All required
	constructor	Required
	toString()	Required
	valueOf()	Required
Number		
	prototype	Required
	MAX_VALUE	Required
	MIN_VALUE	Required
	NaN	Required
	NEGATIVE_INFINITY	–
	POSITIVE_INFINITY	–
	Number([value])	Required
	new Number([value])	Required
Number.prototype		All required
	constructor	Required
	toString([radix])	Required
	valueOf()	Required
Math		
	E	–
	LN10	–
	LN2	–
	LOG2E	–
	LOG10E	–
	PI	–
	SQRT1_2	–
	SQRT2	–
	abs(x)	–
	acos(x)	–
	asin(x)	–
	atan(x)	–
	atan2(y, x)	–
	cos(x)	–

© ABNT 2011 - All rights reserved

Table 5 (continuation)

Object	Method, properties	Operation condition
Math		
	exp(x)	–
	floor(x)	–
	log(x)	–
	max(x, y)	–
	min(x, y)	–
	pow(x, y)	–
	random()	–
	round(x)	–
	sin(x)	–
	sqrt(x)	–
	tan(x)	–
Date		
	prototype	Required
	Date([year, month [, date [, hours [, minutes [,seconds [, ms]]]]]])	Required
	new Date([year, month [, date [, hours [, minutes[, seconds [, ms]]]]]])	Required
	Date(value)	–
	new Date(value)	–
	Date.parse(string)	–
	Date.UTC([year [, month [, date [, hours [,minutes [, seconds [, ms]]]]]]])	–
Date.prototype		
	constructor	Required
	toString()	Required
	valueOf()	–
	getTime()	–
	getYear())	–
	getFullYear()	Required
	getUTCFullYear()	Required
	getMonth()	Required
	getUTCMonth()	Required
	getDate()	Required
	getUTCDate()	Required
	getDay()	Required
	getUTCDay()	Required
	getHours()	Required
	getUTCHours()	Required
	getMinutes()	Required
	getUTCMinutes()	Required
	getSeconds()	Required
	getUTCSeconds()	Required
	getMilliseconds()	Required
	getUTCMilliseconds()	Required
	getTimezoneOffset()	Required
	setTime(time)	–
	setMilliseconds(ms)	Required
	setUTCMilliseconds(ms)	Required

© ABNT 2011 - All rights reserved

Table 5 (continuation)

Object	Method, properties	Operation condition
Date.prototype		
	setSeconds(sec [, ms])	Required
	setUTCSeconds(sec [, ms])	Required
	setMinutes(min [, sec [, ms]])	Required
	setUTCMinutes(min [, sec [, ms]])	Required
	setHours(hour [, min [, sec [, ms]]])	Required
	setUTCHours(hour [, min [, sec [, ms]]])	Required
	setDate(date)	Required
	setMonth(mon [, date])	Required
	setUTCMonth(mon [, date])	Required
	setFullYear(year [, mon [, date]])	Required
	setUTCFullYear(year [, mon [, date]])	Required
	setYear(year)	–
	toLocaleString()	Required
	toUTCString()	Required
	toGMTString()	–

Depending on the middleware implementation, it is possible to have ECMAScript functions mapped to the API provided by Ginga-J, in order to have access to some set-top box resources and Ginga facilities. In this case, the API provided in ECMAScript should follow the same specification presented to Ginga-J imperative environment.

6.3.4 DOM API

The common DOM level 1 API are the following:

— DOMException;

— DOMImplementation;

— DocumentFragment;

— Document;

— Node;

— NodeList;

— NamedNodeMap;

— CharacterData;

— Attr;

— Element;

— Text;

— Comment.

The DOM API, when implemented, should support the common DOM level 1 API for BML for basic services, ACAP-X and DVB-HTML, as follows in Table 6.

© ABNT 2011 - All rights reserved

Table 6 – Common DOM level 1 API

Interface	Attribute/Method	Operation condition
DOMImplementation		
	hasFeature()	Required
Document		
	Doctype	–
	implementation	Required
	documentElement	Required
	createElement()	–
	createDocumentFragment()	–
	createTextNode()	–
	createComment()	–
	createCDATASection()	–
	createProcessingInstruction()	–
	createAttribute()	–
	createEntityReference()	–
	getElementsByTagName()	–
Node		
	nodeName	–
	nodeValue	–
	nodeType	–
	parentNode	Required
	childNodes	–
	firstChild	Required
	lastChild	Required
	previousSibling	Required
	nextSibling	Required
	Attributes	–
	ownerDocument	–
	insertBefore()	–
	replaceChild()	–
	removeChild()	–
	appendChild()	–
	hasChildNodes()	–
	cloneNode()	–
CharacterData		
	Data	Required
	length	Required
	substringData()	–
	appendData()	–
	insertData()	–
	deleteData()	–
	replaceData()	–
Element		
	tagName	Required
	getAttribute()	–
	setAttribute()	–
	removeAttribute()	–
	getAttributeNode()	–
	setAttributeNode()	–
	removeAttributeNode()	–
	getElementsByTagName()	–
	normalize()	–
Text		
	splitText	–

© ABNT 2011 - All rights reserved

7 NCL - XML application declarative language for interactive multimedia presentations

7.1 Modular languages and language profiles

7.1.1 NCL modules

The modularization approach has been used in several W3C language recommendations.

Modules are collections of semantically-related XML elements, attributes and attribute values that represent a unit of functionality. Modules are defined in coherent sets. This coherence is expressed in that the elements of these modules are associated with the same namespace.

NOTE Namespaces are discussed in [Namespaces in XML, 1999].

A language profile is a combination of modules. Modules are atomic, that is, they shall not be subset when included in a language profile. Furthermore, a module specification may include a set of integration requirements, to which language profiles that include the module shall comply.

NCL has been specified in a modular way, allowing the combination of its modules in language profiles. Each profile may group a subset of NCL modules, allowing the creation of languages according to the users' needs. Moreover, NCL modules and profiles can be combined with other language modules, allowing the incorporation of NCL features into those languages and vice-versa.

Commonly, there is a language profile that incorporates nearly all the modules associated with a single namespace. This is the case of the NCL Language profile.

Other language profiles can be specified as subsets of the larger one, or to incorporate a combination of modules associated with different namespaces. Examples of the first case are the Basic DTV and the Enhanced DTV profiles of NCL.

Subsets of the language profile modules used in the definition of the Basic DTV and Enhanced DTV profiles are defined to fit the language to the data television broadcasting environment with its multiple possible presentation devices: television set, mobile devices etc.

NOTE A similar approach is also found in other languages (SMIL 2.1 Specification, 2005] [XHTML 1.0, 2002).

The main purpose of language profile conformance is to enhance interoperability. The mandatory modules are defined in such a way that any document interchanged in a conforming language profile will yield a reasonable presentation. The document formatter, while supporting the associated mandatory module set, should ignore all other (unknown) elements and attributes.

NOTE Document renderer, user agent and player are other names used with the same meaning of document formatter.

The NCL 3.0 edition revises the functionalities contained in NCL 2.3 [NCL Main Profile, 2005], and is partitioned into 15 functional areas, which are partitioned again into modules. From the 15 functional areas, 14 are used to define the Enhanced DTV and the Basic DTV profiles. Two functional areas have modules with the same semantics defined by SMIL 2.0. The 14 used functional areas and their corresponding modules are:

1) Structure

Structure Module

2) Layout

Layout Module

3) Components

© ABNT 2011 - All rights reserved

Media Module

Context Module

4) Interfaces

MediaContentAnchor Module

CompositeNodeInterface Module

PropertyAnchor Module

SwitchInterface Module

5) Presentation Specification

Descriptor Module

6) Linking

Linking Module

7) Connectors

ConnectorCommonPart Module

ConnectorAssessmentExpression Module

ConnectorCausalExpression Module

CausalConnector Module

CausalConnectorFunctionality Module

ConnectorBase Module

8) Presentation Control

TestRule Module

TestRuleUse Module

ContentControl Module

DescriptorControl Module

9) Timing

Timing Module

10) Reuse

Import Module

© ABNT 2011 - All rights reserved

EntityReuse Module

ExtendedEntityReuse Module

11) Navigational Key

KeyNavigation Module

12) Animation

Animation Module

13) Transition Effects

TransitionBase Module

Transition Module

14) Meta-Information

Metainformation Module

7.1.2 Identifiers for NCL 3.0 module and language profiles

Each NCL profile should explicitly state the namespace URI that is to be used to identify it.

Documents authored in language profiles that include the NCL Structure module can be associated with the “application/x-ncl+xml” MIME type. Documents using the “application/x-ncl+xml” MIME type shall be host language conformant.

The XML namespace identifiers for the complete set of NCL 3.0 modules, elements and attributes are contained within the following namespace: http://www.ncl.org.br/NCL3.0/

Each NCL module has a unique identifier associated with it. The identifiers for NCL 3.0 modules shall be in agreement with Table 7.

Modules may also be identified collectively. The following module collections are defined:

— modules used by the NCL 3.0 Language profile: http://www.ncl.org.br/NCL3.0/LanguageProfile;

— modules used by the NCL 3.0 Causal Connector profile: http://www.ncl.org.br/NCL3.0/CausalConnectorProfile;

— modules used by the NCL 3.0 Enhanced DTV profile: http://www.ncl.org.br/NCL3.0/EDTVProfile;

— modules used by the NCL 3.0 Basic DTV profile: http://www.ncl.org.br/NCL3.0//BDTVProfile.

© ABNT 2011 - All rights reserved

Table 7 – The NCL 3.0 module identifiers

Modules	Identifiers
Animation	http://www.ncl.org.br/NCL3.0/Animation
CompositeNodeInterface	http://www.ncl.org.br/NCL3.0/CompositeNodeInterface
CausalConnector	http://www.ncl.org.br/NCL3.0/CausalConnector
CausalConnectorFunctionality	http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality
ConnectorCausalExpression	http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression
ConnectorAssessmentExpression	http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression
ConnectorBase	http://www.ncl.org.br/NCL3.0/ConnectorBase
ConnectorCommonPart	http://www.ncl.org.br/NCL3.0/ConnectorCommonPart
ContentControl	http://www.ncl.org.br/NCL3.0/ContentControl
Context	http://www.ncl.org.br/NCL3.0/Context
Descriptor	http://www.ncl.org.br/NCL3.0/Descriptor
DescriptorControl	http://www.ncl.org.br/NCL3.0/DescriptorControl
EntityReuse	http://www.ncl.org.br/NCL3.0/EntityReuse
ExtendedEntityReuse	http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse
Import	http://www.ncl.org.br/NCL3.0/Import
Layout	http://www.ncl.org.br/NCL3.0/Layout
Linking	http://www.ncl.org.br/NCL3.0/Linking
Media	http://www.ncl.org.br/NCL3.0/Media
MediaContentAnchor	http://www.ncl.org.br/NCL3.0/MediaContentAnchor
KeyNavigation	http://www.ncl.org.br/NCL3.0/KeyNavigation
PropertyAnchor	http://www.ncl.org.br/NCL3.0/PropertyAnchor
Structure	http://www.ncl.org.br/NCL3.0/Structure
SwitchInterface	http://www.ncl.org.br/NCL3.0/SwitchInterface
TestRule	http://www.ncl.org.br/NCL3.0/TestRule
TestRuleUse	http://www.ncl.org.br/NCL3.0/TestRuleUse
Timing	http://www.ncl.org.br/NCL3.0/Timing
TransitionBase	http://www.ncl.org.br/NCL3.0/TransitionBase
Transition	http://www.ncl.org.br/NCL3.0/Transition
Metainformation	http://www.ncl.org.br/NCL3.0/MetaInformation

Three SMIL modules [SMIL 2.1 Specification, 2005] were used as the basis for the NCL Transition module and the NCL Metainformation module definitions. The identifiers of these SMIL 2.0 modules are shown in Table 8.

Table 8 – The SMIL 2.0 module identifiers

Modules	Identifiers
BasicTransitions	http://www.w3.org/2001/SMIL20/BasicTransitions
TransitionModifiers	http://www.w3.org/2001/SMIL20/TransitionsModifiers
Metainformation	http://www.w3.org/2001/SMIL20/Metainformation

© ABNT 2011 - All rights reserved

7.1.3 NCL Version information

The following processing instructions shall be written in an NCL document. They identify NCL documents that contain only the elements defined in this Standard, and the NCL version to which the document conforms.

<?xml version="1.0" encoding="ISO-8859-1"?>

<ncl id="any string" xmlns="http://www.ncl.org.br/NCL3.0/profileName">

The *id* attribute of an <ncl> element may receive any string that·matches·the NCName type [Namespaces in XML:1999] as a value. That is, may receive any string value that begins with a letter or an underscore and that only contains letters, digits, "." and "_".

The version number of an NCL document specification consists of a major number and a minor number, separated by a dot. The numbers are represented as a decimal number character string with leading zeros suppressed. The initial standard version number is 3.0.

New NCL versions shall be released in accordance to the following versioning policy:

— if receivers that conform to older versions can still receive a document based on the revised specification, in relation to error corrections or operational reasons, the new version of NCL shall be released with the minor number updated;

— if receivers that conform to older versions cannot receive a document based on the revised specifications, the major number shall be updated.

A specific version is specified in the URI path http://www.ncl.org.br/NCLx.y/profileName, where the version number “x.y” is written immediately after the “NCL”.

The profileName, in the URI path, shall be EDTVProfile (Enhanced DTV Profile), BDTVProfile (Basic DTV Profile), or CausalConnectorProfile.

7.2 NCL modules

7.2.1 General remarks

The main definitions made by each NCL 3.0 modules that are present in the NCL 3.0 Basic DTV and the Enhanced DTV profiles are given in Sections 7.2.2 to 7.2.15.

The complete definition of these NCL 3.0 modules, using XML Schemas, is presented in Annex A. Any ambiguity found in this text can be clarified by consulting the XML schemas.

After discussing each module, a table is presented indicating the module elements and their attributes. For a given profile, attributes and contents (child elements) of an element may be defined in the module itself or in the language profile that groups the modules. The value of an attribute may not contain quotation marks (“). When a value is a string, it may be any string that·matches·the NCName type [Namespaces in XML:1999]. That is, the value may be any string that begins with a letter or an underscore and that only contains letters, digits, "." and "_".

Therefore, tables in this section show attributes and contents that come from NCL Enhanced DTV profile, besides those defined in the modules themselves. Tables in Section 7.3.3 show the attributes and contents that come from NCL Basic DTV profile, besides those defined in the NCL modules themselves. Element attributes that are required are underlined. In the tables, the following symbols are used: (?) optional (zero or one occurrence), (|) or, (*) zero or more occurrences, (+) one or more occurrences. The child element order is not specified in the tables.

© ABNT 2011 - All rights reserved

7.2.2 Structure functionality

The Structure functionality has just one module, called Structure, which defines the basic structure of an NCL document. It defines the root element, called <ncl>, the <head> element and the <body> element, following the terminology adopted by other W3C standards. The <body> element of an NCL document is treated as an NCM context node (NCMCore:2005).

In NCM, the conceptual data model of NCL, a node may be a context, a switch or a media object. All NCM nodes are represented by corresponding NCL elements. Context nodes, according what ir defined in 7.2.4, contain other NCM nodes and links.

Almost all NCL elements may have the *id* attribute. This attribute may receive any string that matches the NCName type definition [Namespaces in XML: 1999], as its value. That is, may receive any string value that begins with a letter or an underscore and that only contains letters, digits, "." and "_". The *id* attribute univocally identifies an element in an NCL document.

The <ncl> element shall have and the <body> element may have the *id* attribute defined. The *id* attribute uniquely identifies an element within a document. Its value is an XML identifier.

The *title* attribute of <ncl> offers advisory information about the element. Values of the *title* attribute may be rendered by user agents in a variety of ways.

The *xmlns* attribute declares an XML namespace, that is, it declares the primary collection of XML-defined constructs the document uses. The attribute's value is the URL (Uniform Resource Locator), that identifies where the namespace is officially defined. Three values are allowed for the *xmlns* attribute: http://www.ncl.org.br/NCL3.0/EDTVProfile, and http://www.ncl.org.br/NCL3.0/BDTVProfile, for the Enhanced and Basic DTV profiles, respectively, and http://www.ncl.org.br/NCL3.0/CausalConnectorProfile, for the Causal Connector profile. An NCL formatter shall know that the schemaLocation for these namespaces is, by default, respectively:
http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd,
http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd, and
http://www.ncl.org.br/NCL3.0/profiles/NCL30CausalConnector.xsd

Child elements of <head> and <body> are defined in other NCL modules. The order in which the <head> child elements should be declared is: importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*.

The elements of this module, their child elements, and their attributes shall be in agreement with Table 9.

Table 9 – Extended Structure module

Elements	Attributes	Content
ncl	*id, title, xmlns*	(head?, body?)
head		(importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*)
body	*id*	(port| property| media| context| switch| link | meta | metadata)*

© ABNT 2011 - All rights reserved

7.2.3 Layout functionality

The Layout functionality has a single module, named Layout, which specifies elements and attributes that may define how objects will be initially presented inside regions of output devices. Indeed, this module may define initial values for homonym NCL properties defined in <media>, <body>, and <context> elements (see 7.2.4).

A <regionBase> element, which may be declared in the NCL document <head>, defines a set of <region> elements, each of which may contain another set of nested <region> elements, and so on, recursively.

The <regionBase> element may have the *id* attribute, and <region> elements shall have the *id* attribute. As usual, the *id* attribute uniquely identifies the element within a document.

Each <regionBase> element is associated with a class of devices where presentation will take place. In order to identify the association, the <regionBase> element defines the *device* attribute, which may have the values: “systemScreen (i)” or “systemAudio(i)”, where *i* is an integer greater than zero. The chosen class defines global environment variables: system.screenSize(i), system.screenGraphicSize(i), and system.audioType(i), as defined in Table 12 (see 7.2.4). When the attribute is not specified, the presentation shall take place in the same device that runs the NCL formatter.

NOTE 1 There are two different types of device classes: active and passive. In an active class, a device is able to run NCL’s media players. In a passive class, a device is not required to run NCL’s media players, only to exhibit a bit map or a sequence of audio samples received from another device. In SBTVD, systemScreen (1) and systemAudio(1) are reserved to passive classes, and systemScreen (2) and systemAudio(2) are reserved to active classes.

NOTE 2 The <regionBase> element that defines a passive class may also have a *region* attribute. This attribute is used to identify a <region> element in another <regionBase> associated with the active class where the device that creates the bit map sent to the passive-class devices is registered; in the specified region the bit map must also be exhibited.

The interpretation of the region nesting inside a <regionBase> should be made by the software in charge of the document presentation orchestration (thai is, the NCL *formatter*).For the purposes of this Standard, a first nesting level shall be interpreted as defining the device area where the presentation would take place; the second nesting level as windows (for example, presentation areas in the screen) of the parent area; and the other levels as regions inside these windows.

A <region> can also define the following attributes: *title*, *left*, *right, top, bottom*, *height*, *width*, and *zIndex*. All these attributes have the usual meaning.

The position of a region, as specified by its *top*, *bottom*, *left*, and *right* attributes, is always relative to the parent geometry, which is defined by the parent <region> element or the total device area in the case of first nesting level regions. Attribute values may be non-negative “percentage” values, or integer pixel units. For pixel values, the author may omit the “px” unit qualifier (for example, “100”). For percentage values, on the other hand, the “%” symbol shall be indicated (for example, “50%”). The percentage is always relative to the parent’s width, in the case of *right*, *left* and *width* definitions, and parent’s height, in the case of *bottom*, *top* and *height* definitions.

The *top* and *left* attributes are the primary region positioning attributes. They place the left-top corner of the region in the specified distance away from the left-top edge of the parent region (or the device left-top edge in the case of the outermost region). Sometimes, explicitly setting the *bottom* and *right* attributes can be helpful. Their values state the distance between the region’s right-bottom corner and the right-bottom corner of the parent region (or the device right-bottom edge in the case of the outermost region) (see Figure 2).

© ABNT 2011 - All rights reserved

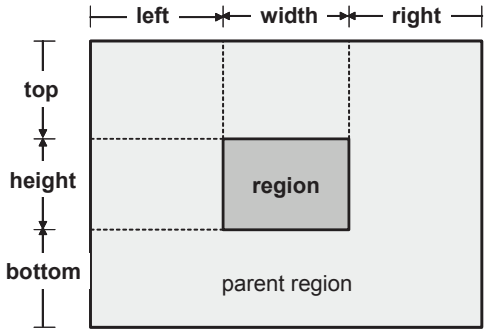

Figure 2 – Region positioning attributes

Regarding region sizes, when they are specified by declaring *width* and *height* attributes using the "%" notation, the size of the region is relative to the size of its parent geometry as aforementioned. Sizes declared as absolute pixel values maintain those absolute values. The intrinsic size of a region is equal to the size of the logical parent's geometry. This means that, if a nested region doesn't specify any positioning or size values, it shalll be assumed to have the same position and size values of its parent region. In particular, when a first level region doesn't specify any positioning or size values, it shall be assumed to be the whole device presentation area.

When the user specifies *top*, *bottom* and *height* information for the same <region>, spatial inconsistencies can occur. In this case, the *top* and *height* values shall have precedence over the *bottom* value. Analogously, when the user specifies inconsistent values for the *left*, *right* and *width* <region> attributes, the *left* and *width* values shall be used to compute a new *right* value. When any of these attributes is not specified and cannot have its value computed from the other attributes, this value shall be inherited from the corresponding parent value. Another restriction is that child regions cannot stay outside the area established by their parent regions.

The *zIndex* attribute specifies the region superposition precedence. Regions with greater *zIndex* values shall be stacked on top of regions with smaller *zIndex* values. If two presentations generated by elements A and B have the same stack level then, if the display of an element B starts later than the display of an element A, the presentation of B shall be stacked on top of the presentation of A (temporal order); otherwise, if the display of the elements starts at the same time, the stacked order is chosen arbitrarily by the formatter. When not specified, the *zIndex* attribute is set equal to 0 (zero).

The Layout module also defines the *region* attribute that is used by a <descriptor> element (see 7.2.6) to refer a Layout <region> element.

The elements of this module, their child elements and their attributes shall be in agreement with Table 10.

Table 10 – Extended Layout module

Elements	Attributes	Content
regionBase	*d, device, region*	(importBase|region)+
region	*id, title, left, right, top, bottom, height, width, zIndex*	(region)*

7.2.4 Components functionality

The Components functionality is partitioned into two modules, called Media and Context.

The Media module defines basic media object types. For defining media objects, this module defines the <media> element. Each media object has two main attributes, besides its *id* attribute: *src*, which defines a URI of the object content, and *type*, which defines the object type.

© ABNT 2011 - All rights reserved

The URI (Uniform Resource Identifier) shall be in agreement with Table 11.

Table 11 – Allowed URI

Scheme	Scheme-specific-part	Use
file:	///file_path/#fragment_identifier	For local files
http:	//server_identifier/file_path/#fragment_identifier	For remote files downloaded from the interactive channel using the http protocol
https:	//server_identifier/file_path/#fragment_identifier	For remote files downloaded from the interactive channel using the https protocol
rtsp:	//server_identifier/file_path/#fragment_identifier	for streams downloaded from the interactive channel using the rtsp protocol
rtp:	//server_identifier/file_path/#fragment_identifier	For streams downloaded from the interactive channel using the rtp protocol
ncl-mirror:	//media_element_identifier	For a content flow identical to the one in presentation by another media element
sbtvd-ts:	//program_number.component_tag	For elementary streams received from the transport stream

An absolute URI by itself contains all information needed to locate its resource. Relative URI are also allowed. Relative URI are incomplete addresses that are applied to a base URI to complete the location. The portions omitted are the URI scheme and server, and potentially part of URI path, as well.

The primary benefit of using relative URI is that documents and directories containing them may be moved or copied to other locations without requiring changing the URI attribute values within the documents. This is especially interesting when transporting documents from the server part (usually broadcasters) to the receivers. Relative URI paths are typically used as a short means of locating media files stored in the same directory as the current NCL document, or in a directory close to it. They often consist of just the filename (optionally with a fragment identifier into the file). They may also have a relative directory path before the filename.

It should be emphasized that references to streaming video or audio resources may not cause tuning to occur. References that imply tuning to access a resource shall behave as if the resource were unavailable.

The *type* attribute's allowed values depend on the NCL profile and shall follow MIME Media Types format (or, more simply, mimetypes). A mimetype is a character string that defines the class of media (audio, video, image, text, application) and a media encoding type (such as jpeg, mpeg etc.). Mimetypes may be registered or informal. Registered mimetypes are controlled by the IANA (Internet Assigned Numbers Authority). Informal mimetypes are not registered with IANA, but are defined by common agreement; they usually have an "x-"before the media type name.

Five special types are defined: application/x-ginga-NCL (or application/x-ncl-NCL); application/x-ginga-NCLua (or application/x-ncl-NCLua), application/x-ginga-NCLet (or application/x-ncl-NCLet), application/x-ginga-settings (or application/x-ncl-settings), and application/x-ginga-time (or application/x-ncl-time).

The "application/x-ginga-NCL" type shall be applied to <media> elements with NCL code content (indeed, an NCL application can embed another NCL application). The application/x-ginga-NCLua type shall be applied to <media> elements with Lua imperative code content (See Section 10). The application/x-ginga-NCLet type shall be applied to <media> elements with Xlet imperative code content (See Section 11).

The application/x-ginga-settings shall be applied to a special <media> element (there may be only one in an NCL document) whose properties are global variables defined by the document author or reserved environment variables that may be manipulated by the NCL document processing. Table 12 states the already defined variables and their semantics.

© ABNT 2011 - All rights reserved

Table 12 – Environment variables

Group	Variable	Semantics	Possible values
system • set of variables managed by the receiver system; • they may be read, but they may not have their values changed by an NCL application, a Lua procedure or an Xlet procedure; • receiver's native applications may change the variables' values; • they shall persist during all receiver life cicle.	system.language	Audio language	ISO 639-1 code
	system.caption	Caption language	ISO 639
	system.subtitle	Subtitle Language	ISO 639
	system.returnBitRate(i)	The whole bit rate of the interactive channel (i) in Kbps	real
	system.screenSize	Device screen size, in (lines, pixels/line), when a class is not defined	(integer, integer)
	system.screenGraphicSize	Resolution set for the device's screen graphics plane, in (lines, pixels/line), when a class is not defined	(integer, integer)
	system.audioType	Type of the device audio, when a class is not defined	"mono" | "stereo" | "5.1"
	system.screenSize (i)	Screen size of the class (i) of devices, in (lines, pixels/line)	(integer, integer)
	system.screenGraphicSize (i)	Resolution set for the screen graphics plane of the class (i) of devices, in (lines, pixels/line)	(integer, integer)
	system.audioType(i)	Type of the audio of the class (i) of devices	"mono" | "stereo" | "5.1"
	system.devNumber(i)	Number of exhibition devices registered in the class (i)	integer
	system.classType(i)	Type of the class (i)	("passive" | "ative")
	system.info(i)	List of class (i)'s media players	string
	system.classNumber	Number of classes that has been defined	integer
	system.CPU	CPU performance in MIPS, regarding its capacity to run applications	real
	system.memory	Minimum memory space in Mbytes provided to applications	integer
	system.operatingSystem	Type of the Operating System	string to be defined
	system.javaConfiguration	Java configuration type and version supported by the receiver JVM	string (type immediately followed by version, as for example: "CLDC1.1")
	system.javaProfile	Java profile type and version supported by the receiver JVM	string (type immediately followed by version, as for example: "MIDP2.0")
	system.luaVersion	Version of the Lua engine supported by the receiver	string
	system.ncl.version	NCL language version	string
	system.GingaNCL.version	Ginga-NCL environment version	string
	system.GingaJ.version	Ginga-J environment version	string
	system.xxx	Any variable with the "system" prefix shall be reserved for future use	

© ABNT 2011 - All rights reserved

Table 12 (continuation)

Group	Variable	Semantics	Possible values
user • set of variables managed by the receiver system; • they may be read, but they may not have their values changed by an NCL application, a Lua procedure or an Xlet procedure; • receiver's native applications may change the variables' values; • they shall persist during all receiver life cicle	user.age	User age	integer
	user.location	User location	string
	user.genre	User genre	"m"| "f"
	user.xxx	Any variable with the "user" prefix shall be reserved for future use	
default • set of variables managed by the receiver system; • they may be read and have their values changed by an NCL application, a Lua procedure or an Xlet procedure; • receiver's native applications may change the variables' values; • they shall persist during all receiver life cicle, however, they shall be set to their initial values when a new channel is tunned.	default.focusBorderColor	Default color applied to the border of an element in focus	"white" | "black" | "silver" | "gray" | "red" | "maroon" | "fuchsia" | "purple" | "lime" | "green" | "yellow" | "olive" | "blue" | "navy" | "aqua" | "teal"
	default.selBorderColor	Default color applied to the border of an element in focus when activated	"white" | "black" | "silver" | "gray" | "red" | "maroon" | "fuchsia" | "purple" | "lime" | "green" | "yellow" | "olive" | "blue" | "navy" | "aqua" | "teal"
	default.focusBorderWidth	Default width (in pixels) applied to the border of an element in focus	integer
	default.focusBorderTransparency	Default transparency applied to the border of an element in focus	a real value between 0 and 1, or a real value in the range [0,100] ending with the character "%" (for example 30 %), with "1" or "100 %" meaning full transparency and "0" or "0 %" meaning no transparency
	default.xxx	Any variable with the "default" prefix shall be reserved for future use	

© ABNT 2011 - All rights reserved

Table 12 (continuation)

Group	Variable	Semantics	Possible values
service • set of variables managed by the NCL formatter; • they may be read and have their values changed by an NCL application of the same service; • they may be read but they may not have their values changed by a Lua procedure or an Xlet procedure of the same service; variable changes shall be done using NCL commands; • they shall persist at least during the service life cicle.	service.currentFocus	The focusIndex value of the <media> element on focus	integer
	service.currentKeyMaster	Identifier (id) of the <media> element that controls the navigational keys; if the <media> element is not being presented or is not paused, the navigational key control pertains to the NCL Formatter	string
	service.xxx	Any variable with the “service” prefix shall follow the rules specified for the group	
si • set of variables managed by the middleware; • they may be read, but they may not have their values changed by an NCL application, a Lua procedure or an Xlet procedure; • they shall persist at least until the next channel tuning.	si.numberOfServices	Number of services available in the country for the tuned channel. NOTE The value for this variable should be obtained from the number of PMT tables specified in the PAT table of the transport stream received in the tuning channel (see ISO/IEC 13818-1:2007). The variable value should take into account only the PMT tables whose fields country_code, specified in the country_availability_descriptor (See Section 8.3.6 of ABNT NBR 15603-2:2007) related with the table, are equivalent to the value of the *user.location* variable of the Settings node.	integer
	si.numberOfPartialServices	Number of 1-seg services available in the country for the tuned channel. NOTE The value for this variable should be obtained from the number of PMT tables specified in the PAT table of the transport stream received in the tuning channel (see ISO/IEC 13818-1:2007). The variable value should take into account only the PMT tables whose country_code fields, specified in the country_availability_descriptor (See Section 8.3.6 of ABNT NBR 15603-2:2007) related with the table, are equal to the value of the *user.location* variable of the Settings node, and whose program_number fields are equivalent to the service_id fields of the partial_reception_descriptor related with the NIT tables.	integer

© ABNT 2011 - All rights reserved

Table 12 (continuation)

Group	Variable	Semantics	Possible values
	si.channelNumber	Number of the tuned channel. NOTE The value for this variable should be obtained from the remote_control_key_id filed of the ts_information_descriptor (see Section 8.3.42 of ABNT NBR 15603-2:2007) of the NIT table (see Section 7.2.4 of ABNT NBR 15603-2:2007) that describes the current service.	integer
	si.xxx	Any variable with the “si” prefix shall follow the rules specified for the group	
channel • set of variables managed by the NCL formatter; • they may be read and have their values changed by an NCL application of the same channel; • they may be read but they may not have their values changed by a Lua procedure or an Xlet procedure of the same channel; variable changes shall be done using NCL commands; • they shall persist at least till the next channel tuning.	channel.keyCapture	Request of alphanumeric keys for NCL applications	(string)
	channel.virtualKeyboard	Request of a virtual keyboard for NCL applications	(true | false)
	channel.keyboardBounds	Virtual keyboard region (left, top, width, height)	(integer, integer, integer, integer)
	channel.xxx	Any variable with the “channel” prefix shall follow the rules specified for the group	
shared • set of variables managed by the NCL formatter; • they may be read and have their values changed by an NCL application; • they may be read but they may not have their values changed by a Lua procedure or an Xlet procedure; variable changes shall be done using NCL commands; • they shall persist at least during the life cicle of the service that has defined them.	shared.xxx	Any variable with the “shared” prefix shall follow the rules specified for the group	

NOTE 1 The media object of application/x-ginga-settings type does not have content to be exhibited. Once an NCL application is started, the properties of this media object are available for rule evaluations defined by <rule> elements. In order to use these properties in link definitions, they must be explicitly declared.

NOTE 2 The content of a <media> element of application/x-ginga-time type is specified according with the following syntax: Year“:”Month“:”Day“:”Hours“:”Minutes“:”Seconds“.”Fraction, where Year is an integer; Month is an integer in the [1,12] interval; Day is an integer in the [1,31] interval; Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; Fraction is a positive integer. When it is started, the UTC content is not exhibited, however, it can be used to define content anchors by using <area> elements.

The application/x-ginga-time type shall be applied to a special <media> element (it may be only one in an NCL document), whose content is the Universal Time Coordinated (UTC). Note that any continuous <media> element with no source can be used to define a clock relative to the <media> element start time.

© ABNT 2011 - All rights reserved

Table 13 shows some possible values of the *type* attribute for the Enhanced DTV and Basic DTV profiles and the associated file extensions. The required types are defined in ABNT NBR 15601. The *type* attribute is optional (except for <media> elements with no *src* attribute defined) and should be used to guide the player (presentation tool) choice by the formatter. When the *type* attribute is not specified, the formatter should use the content extension specification in the *src* attribute to make the player choice.

When there is more than one player for the type supported by the formatter, the *player* property of the <media> *element* may specify which one will be used for presentation. Otherwise the formatter shall use a default player for that type of media.

Table 13 – MIME Media types for Ginga-NCL formatters

Media type	File extensions
text/html	htm, html
text/plain	txt
text/css	css
text/xml	xml
image/bmp	bmp
image/png	png
image/gif	gif
image/jpeg	jpg, jpeg
audio/basic	wav
audio/mp3	mp3
audio/mp2	mp2
audio/mpeg	mpeg, mpg
audio/mpeg4	mp4, mpg4
video/mpeg	mpeg, mpg
application/x-ginga-NCL	ncl
application/x-ginga-NCLua	lua
application/x-ginga-NCLet	class, jar
application/x-ginga-settings	*no src (source)*
application/x-ginga-time	*no src (source)*

The Context module is responsible for the definition of context nodes through <context> elements. 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. As usual, the *id* attribute uniquely identifies each <context> and <media> element within a document.

The *instance*, *refer* and *descriptor* attributes are extensions defined in other modules and are discussed in the definition of those modules.

A <media> element of application/x-ginga-NCL type may not have the *instance* and *refer* attributes.

The elements of these two modules, their child elements, and their attributes shall be in agreement with Tables 14 and 15.

© ABNT 2011 - All rights reserved

Table 14 – Extended Media module

Elements	Attributes	Content
media	*id, src, refer, instance, type, descriptor*	(area|property)*

Table 15 – Extended Context module

Elements	Attributes	Content
context	*id, refer*	(port|property|media|context|link|switch|meta|metadata)*

7.2.5 Interfaces functionality

The Interfaces functionality allows for the definition of node (media object or composite objects) interfaces that will be used in relationships with other node interfaces. This functionality is partitioned into four modules:

— MediaContentAnchor, which allows for content anchor (or area) definitions for media nodes (<media> elements);

— CompositeNodeInterface, which allows for port definitions for composite nodes (<context> and <switch> elements);

— PropertyAnchor, which allows for the definition of node properties as node interfaces; and

— SwitchInterface, which allows for the definition of special interfaces for <switch> elements.

The MediaContentAnchor module defines the <area> element, which , allows for the definition of content anchors representing spatial portions, through the *coords* attribute (as in XHTML); the definition of content anchors representing temporal portions, through *begin* and *end* attributes; and the definition of content anchors representing temporal and spatial portions through *coords*, *begin* and *end* attributes. In addition, the <area> element allows for the definition of textual anchors, through the *beginText*, *beginPosition* and *endText*, *endPosition* attributes that define the string and the string's occurrence in the text, respectively. Besides, the <area> element may also define a content anchor based on the number of audio samples or video frames, through *first* and *last* attributes, which shall indicate the initial and final sample/frame. Moreover, the <area> element may also define a content anchor based on the *label* attribute, which specifies a string that should be used by the media player to identify a content region. Additionally, the <area> element may define a content anchor using the *clip* attribute, which specifies a triple value that shall be used by the media player to identify a clip in the content of a declarative hypermedia object.

If the *begin* attribute is defined, but the *end* attribute is not specified, the end of the whole media content presentation shall be assumed as the anchor ending. On the other hand, if the *end* attribute is defined, but without an explicit *begin* definition, the start of the whole media content presentation shall be considered as the anchor beginning. Analogous behavior is expected from the *first* and *last* attributes. In the case of a <media> element of the application/x-ginga-time type, the *begin* and *end* attributes shall be always defined and shall assume an absolute value of the Universal Time Coordinated (UTC). In textual content anchors, if the end of the anchor region is not defined, the end of the text content shall be assumed. If the beginning of the content anchor region is not defined, the beginning of the text content shall be assumed.

Except for the <media> element of the application/x-ginga-time type, the *begin* and end attributes shall be specified according with one of the following syntax:
a) Hours":"Minutes":"Seconds"."Fraction, Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; Fraction is a positive integer;
b) Seconds"s", where Seconds is a positive real number.

For the <media> element of the application/x-ginga-time type, the *begin* and end attributes shall be specified according with the following syntax: Year":"Month":"Day":"Hours":"Minutes":"Seconds"."Fraction, according to the country time zone. The NCL user agent is responsible for translating the value for the country time zone to the one corresponding to the UTC.

© ABNT 2011 - All rights reserved

As usual, <area> elements shall have the *id* attribute, which uniquely identifies the element within a document.

The <area> element and its attributes shall be in agreement with Table 16.

Table 16 – Extended MediaContentAnchor module

Elements	Attributes	Content
area	*id, coords, begin, end, beginText, beginPosition endText, endPosition, first, last, label, clip*	empty

The CompositeNodeInterface module defines the <port> element, which specifies a composite node port with its respective mapping to an interface (*interface* attribute) of one of its components (specified by the *component* attribute).

NOTE The *first* and *last* attributes are specified according with one of the following syntax:
a) Samples"s", where Samples is a positive integer;
b) Frames"f", where Frames is a positive integer;
c) NPT"npt", where NPT is the Normal Play Time value.

In NCM, every node (media or context node) shall have an anchor with a region representing the whole content of the node. This anchor is called the *whole content anchor* and is declared by default in NCL documents. Except for media objects with imperative code content (<media type="application/x-ginga-NCLua" …>, for example), every time an NCL component is referred without specifying one of its anchors, the *whole content anchor* is assumed.

The <port> element and its attributes shall be in agreement with Table 17.

Table 17 – Extended CompositeNodeInterface module

Elements	Attributes	Content
port	*id, component, interface*	empty

The PropertyAnchor module defines an element named <property>, which may be used for defining a node property or a group of node properties as one of its interfaces (anchors). The <property> element defines the *name* attribute, which indicates the name of the property or property group, and the *value* attribute, an optional attribute that defines an initial value for the *name* property. The parent element shall not have <property> elements with the same *name* attribute values.

It is possible to have NCL document players (formatters) that define some node properties as node interfaces, implicitly. However, in general, it is a good practice to explicitly define the interfaces.

The <body>, <context>, and <media> elements may have several embedded properties, which are not explicitly declared by <property> elements. Examples of these properties can be found among those that define the media object placement during a presentation, the presentation duration, and others that define additional presentation characteristics: top, left, bottom, right, width, height, plan, explicitDur, background, transparency, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, fontColor, fontFamily, fontStyle, fontSize, fontVariant, fontWeight, reusePlayer, playerLife, etc. These properties assume as their initial values those defined in homonym attributes of their node-associated descriptor and region (see 7.2.3 e 7.2.6). Some properties have their values defined by the middleware system, as for example, the *contentId* property (associated to a continuous-media object whose content is defined referring to an elementary stream), which has "null" as its initial value and is set to the identifier value transported in the NPT reference descriptor (in a field of the same name: contentId), as soon as the associated continuous-media object is started. Another example is the *standby* property that shall be set to "true" while an already started continuous-media object content referring to an elementary stream is temporarily interrupted by another interleaved content, in the same elementary stream. However, in any case, when an embedded property is used in a relationship, it shall be explicitly declared as a <property> (interface) element. Therefore, each property has a hiden Boolean attribute named *externable*, which is defined as "false" by *default* and as "true"when the property is explicitly declared by a <property> element.

© ABNT 2011 - All rights reserved

NOTE 1 Properties (and their initial values) of an NCL object may be defined using only <property> elements. The <descriptor>, <descriptorParam>, and <region> elements are only additional options for defining initial values for properties.

NOTE 2 The *standby* property may be set to "true" when the identifier value transported in the NPT reference descriptor (in a field of the same name: contentId) signalized as non-paused is different from the *contentId* property value.

NOTE 3 The *visible* property may also be associated with a <context> or <body> element. In these cases, when the property's value is equal to "true", the *visible* property of each child element of the composition shall be taken into account. When the property's value is equal to "false", all child elements of the composition shall be exhibited but hidden. In particular, when a document has its <body> element with its *visible* property set to "false" and its presentation event in the *paused* state, the document is said to be in stand-by. When an application is in stand-by, the service's main video shall be dimensioned to 100 % of the screen, and the main audio shall be set to 100 % of volume.

A group of node properties may also be explicitly declared as a single <property> (interface) element, allowing authors to specify the value of several properties within a single property. The following groups shall be recognized by an NCL formatter: *location*, grouping (left, top), in this order; *size*, grouping (width, height), in this order; and *bounds*, grouping (left, top, width, height), in this order.

When a formatter treats a change in a property group it shall only test the process consistency at its end. The words *top, left, bottom, right, width, height, explicitDur, background, transparency, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, fontColor, fontFamily, fontStyle, fontSize, fontVariant, fontWeight, reusePlayer, playerLife*, *location, size* and *bounds* are reserved words for values of the *name* attribute of the <property> element.

The <property> element and its attributes shall be in agreement with Table 18.

Table 18 – Extended PropertyAnchor module

Elements	Attributes	Content
property	*name, value*	empty

The SwitchInterface module allows the creation of <switch> element interfaces (see 7.2.4), which may be mapped to a set of alternative interfaces of internal nodes, allowing a link to anchor on the component chosen when the <switch> is processed (NCM Core:2005). This module introduces the <switchPort> element, which contains a set of *mapping* elements. A *mapping* element defines a path from the <switchPort> to an interface (*interface* attribute) of one of the switch components (specified by its *component* attribute).

It is important to remark that every element representing an object interface (<area>, <port>, <property>, and <switchPort>) shall have an identifier (*id* attribute or *name* attribute).

The <switchPort> element, its child elements and its attributes shall be in agreement with Table 19.

Table 19 – Extended SwitchInterface module

Elements	Attributes	Content
switchPort	*id*	mapping+
mapping	*component, interface*	empty

© ABNT 2011 - All rights reserved

7.2.6 Presentation Specification functionality

The Presentation Specification functionality has a single module named Descriptor. The purpose of this module is to specify temporal and spatial information needed to present each document component. This information is modeled by *descriptor* objects.

The Descriptor module allows for the definition of <descriptor> elements, which contain a set of optional attributes, grouping temporal and spatial definitions, which should be used according to the type of object to be presented. 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. The <descriptor> element shall have the *id* attribute and the <descriptorBase> element may have the *id* attribute, which, as usual, uniquely identifies the elements within a document.

A <descriptor> element may have temporal attributes: *explicitDur* and *freeze,* defined by the Timing module (see 7.2.10); an attribute named *player*, which identifies the presentation tool to be used; an attribute named *region*, which refers to a region defined by elements of the Layout module (see 7.2.3) and key-navigation attributes: *moveLeft*, *moveRight*, *moveUp*; *moveDown*, *focusIndex*, *focusBorderColor*; *focusBorderWidth*; *focusBorderTransparency*, *focusSrc*, *selBorderColor*, and *focusSelSrc*, defined by the KeyNavigation module (see 7.2.12) and transition attributes: *transIn* and *transOut* (see 7.2.14).

NOTE A <descriptor> element of a <media> element of application/x-ginga-NCL type may not have the player attribute. In this case, an NCL player in a specific exhibition device is defined.

A <descriptor> element may also have <descriptorParam> child elements, which are used to parameterize the presentation control of the object associated with the descriptor element. These parameters can, for example, redefine some attribute values defined by the region attributes. They can also define new attributes such as *plan*, defining in which plan of a structured screen an object will be placed; *rgbChromakey*, defining the RGB color to be set as transparent, *background*, specifying the background color used to fill the area of a region displaying media that is not filled by the media itself; *visible,* allowing the object presentation to be seen or hidden; *fit*, indicating how an object will be presented; *scroll*, which allows the specification of how an author would like to configure the scroll in a region; *transparency*, indicating the degree of transparency of an object presentation; *style*, which refers to a style sheet [Cascading Style Sheets, 1998] with information for text presentation, for example; and also specific attributes for audio objects, such as *soundLevel*, *balanceLevel*, *trebleLevel* and *bassLevel*. Besides, <descriptorParam> child elements can determine if a new player shall be instantiated or if a player already instantiated shall be used (*reusePlayer*), and specify what will happen to the player instance at the end of the presentation (*playerLife*). The words top, left, bottom, right, width, height, explicitDur, location, size, bounds, background, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, reusePlayer, and playerLife are reserved words for values of the *name* attribute of the <descriptorParam> element. Some possible values for the reserved parameter/attribute names are presented in Table 20.

© ABNT 2011 - All rights reserved

Table 20 – Reserved parameter/attribute and possible values

Parameter/attribute name	Value
top, left, bottom, right, width, height	A real number in the range [0,100] ending with the character “%” (for example, 30 %), or an integer value specifying the attribute in pixels (a non-negative integer, in the case of width and height)
location	Two numbers separated by comma, each one one following the value rule specified for left and top parameters, respectively
size	Two values separated by comma. Each value shall follow the same rule specified for width and height parameters, respectively
bounds	Four values separated by comma. Each value shall follow the same rule specified for left, top, width and height parameters, respectively
background	Reserved color names: “white”, “black”, “silver”, “gray”, red”, “maroon”, fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua” or “teal”. Another option to specify the color value is stated in ABNT NBR 15606-1. The background value may also be the reserved value “transparent”. This can be helpful to present transparent images, like transparent GIF, superposed on other images or videos. When not specified, the background attribute will take the default value “transparent”
visible	“true” or “false”. Default value = “true”
transparency	A real number in the range [0,1] or a real number in the range [0,100] ending with the character “%” (for example, 30 %), specifying the degree of transparency of an object presentation (“1” or “100 %” means full transparency and “0” or “0 %” means opaque)
fit	“fill”, “hidden”, “meet”, “meetBest”, “slice”. “fill”: scale the object's media content so that it touches all edges of the box defined by the object’s width and height attributes “hidden”: if the intrinsic height/width of the media content is smaller than the height/width attribute, the object shall be rendered starting from the top/left edge and have the remaining height/width filled up with the background color; if the intrinsic height/width of the media content is greater than the height /width attribute, the object shall be rendered starting from the top (left) edge until the height/width defined in the attribute is reached, and have the part of the media content below (to right of) the height (width) clipped “meet”: scale the visual media object while preserving its aspect ratio until its height or width is equal to the value specified by the height or width attributes. The media content left-top corner is positioned at the top-left coordinates of the box; the empty space at the right or the bottom shall be filled up with the background color “meetBest”: the semantic is identical to “meet” except that the image is not scaled greater than 100 % in either dimension “slice”: scale the visual media content while preserving its aspect ratio until its height or width are equal to the value specified in the height and width attributes and the defined presentation box is completely filled. Some parts of the content may get clipped. Overflow width is clipped from the right of the media object. Overflow height is clipped from the bottom of the media object
scroll	“none”, “horizontal”, “vertical”, “both” or “automatic”
style	The locator of a stylesheet file
soundLevel, balanceLevel, trebleLevel, bassLevel	A real number in the range [0, 1] or a real number in the range [0,100] ending with the character “%” (for example, 30 %)
zIndex	An integer number in the range [0, 255], where regions with greater *zIndex* values are stacked on top of regions with smaller *zIndex* values

© ABNT 2011 - All rights reserved

Table 20 (continuation)

Parameter/attribute name	Value
fontColor	Sets the font color (“white”, “black”, “silver”, “gray”, red”, “maroon”, fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal”)
fontFamily	A prioritized list of font family names and/or generic family names
fontStyle	Sets the style of the font (“normal”,or “italic”)
fontSize	The size of a font
fontVariant	Displays text in a “small-caps” font or a “normal” font
fontWeight	Sets the weight of a font (“normal”, or “bold”)
reusePlayer	Boolean value: “false”, “true”. Default value = “false”
playerLife	“keep”, “close”. Default value = “close”

Besides all aforementioned attributes, the <descriptor> element may also have attributes defined in the Transition Effects functionality (see 7.2.14).

NOTE If several values are specified for the same property, the value defined in a <property> element has precedence over the one defined in a <descriptorParam> element, which has precedence over the value defined in an attribute of the corresponding <descriptor> element (including the *region* attribute).

Besides the <descriptor> element, the Descriptor module defines a homonym attribute, which refers to an element of the document descriptor set. When a language profile uses the Descriptor module, it shall determine how *descriptors* will be associated with document components. Following NCM directives, this Standard establishes that the *descriptor* attribute is associated with any media node through <media> elements and through link endpoints (<bind> elements) (see 8.2.1).

The set of descriptors of a document may contain <descriptor> elements or <descriptorSwitch> elements, which allow specifying alternative descriptors (see 7.2.9).

The elements of the Descriptor module, their child elements and their attributes shall be in agreement with Table 21.

Table 21 – Extended Descriptor module

Elements	Attributes	Content
descriptor	*id, player, explicitDur, region, freeze, moveLeft, moveRight, moveUp, moveDown, focusIndex, focusBorderColor, focusBorderWidth, focusBorderTransparency, focusSrc,focusSelSrc, selBorderColor, transIn, transOut*	(descriptorParam)*
descriptorParam	*name, value*	empty
descriptorBase	*id*	(importBase|descriptor|descriptorSwitch)+

7.2.7 Linking functionality

The Linking functionality defines the Linking module, responsible for defining links using connectors. A <link> element may have an *id* attribute, which uniquely identifies the element within a document, and shall have an *xconnector* attribute, which refers to a hypermedia connector URI. The reference shall have the format: *alias#connector_id*, or *documentURI_value#connector_id*, for connectors defined in an external document (see 7.2.11); or simply *connector_id*, for connectors defined in the document itself.

© ABNT 2011 - All rights reserved

The <link> element contains child elements called <bind> elements, which allow to associate nodes with connector roles (see 7.2.8). In order to make this association, a <bind> element has four basic attributes. The first one is called *role*, which is used for referring to a connector role. The second one is called *component*, which is used for identifying the node. The third is an optional attribute called *interface*, used for making reference to the node interface. The fourth is an optional attribute called *descriptor*, used to refer to a descriptor to be associated with the node, as defined by the Descriptor module (see 7.2.6).

NOTE The *interface* attribute may refer to any node interface, that is, an anchor, a property or a port, if it is a composite node. The interface attribute is optional. When it is not specified, the association will be done with the whole node content (see 7.2.5).

If the connector element defines parameters (see 7.2.8), the <bind> or <link> elements should define parameter values, through child elements called <bindParam> and <linkParam>, respectively, both with *name* and *value* attributes. In this case the *name* attribute shall refer to the name of a connector parameter while the *value* attribute shall define a value to be assigned to the respective parameter.

The elements of the linking module, their attributes, and their child elements shall be in agreement with Table 22.

Table 22 - Extended Linking module

Elements	Attributes	Content
bind	*role, component, interface, descriptor*	(bindParam)*
bindParam	*name, value*	empty
linkParam	*name, value*	empty
link	*id, xconnector*	(linkParam*, bind+)

7.2.8 Connectors functionality

The NCL 3.0 Connectors functionality is partitioned into seven basic modules: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression, ConnectorTransitionAssessment, CausalConnector, ConstraintConnector (it does not considered in this Standard), ConnectorBase and CompositeConnector (it also does not considered in this Standard).

The Connectors functionality modules are totally independent from the other NCL modules. These modules are the core by themselves of an XML application language (indeed other NCL 3.0 profiles) for the definition of connectors, which may be used to specify spatio-temporal synchronization relations, treating reference (user interaction) relations as a particular case of temporal synchronization relations.

Besides the basic modules, the Connectors functionality also defines modules that group sets of basic modules, in order to make it easy to define a language profile. This is the case of the CausalConnectorFunctionality module, used in the definition of the EDTV, BDTV and CausalConnector profiles. The CausalConnectorFunctionality module groups the following modules: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression, and CausalConnector.

A <causalConnector> element represents a causal 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 <causalConnector> specifies a relation independently of relationships, that is, it does not specify which nodes (represented by <media>, <context>, <body>, and <switch> elements) will interact through the relation. A <link> element, in its turn, represents a relationship, of the type defined by its connector, interconnecting different nodes. Links representing the same type of relation, but interconnecting different nodes, may reuse the same connector, reusing all previous specifications. A <causalConnector> specifies, through its child elements, a set of interface points, called *roles*. A <link> element refers to a <causalConnector> and defines a set of binds (<bind> child elements of the <link> element), which associate each link endpoint (node interface) to a role of the used connector.

© ABNT 2011 - All rights reserved

Relations in NCL are based on events. An event is an occurrence in time that may be instantaneous or have a measurable duration. NCL 3.0 defines the following types of events:

— *presentation event*, which is defined by the presentation of a subset of the information units of a media object, specified in NCL by the <area> element, or by the media node itself (whole content presentation). Presentation events may also be defined on composite nodes (represented by a <body>, <context>, or <switch> element), representing the presentation of the information units of any node inside a composite node;

— *selection event*, which is defined by the selection of a subset of the information units of a media object being presented, specified in NCL by the <area> element, or by the media node itself (whole content presentation);

— *attribution event*, which is defined by the attribution of a value to a property of a node (represented by a <media>, <body>, <context>, or <switch> element), which shall be declared in a <property> child element of the node; and

— composition event, which is defined by the presentation of the structure of a composite node (represented by a <body>, <context>, or <switch> element). Composition events are used to present the composite map (composite organization). This functionality is optional.

Each event defines a state machine that should be maintained by the NCL formatter, as demonstrated in Figure 3. Moreover, every event has an associated attribute, named *occurrences*, which counts how many times the event transits from occurring to sleeping state during a document presentation. Events of presentation and attribution types have also an attribute named *repetitions*, which counts how many times the event shall be automatically restarted (transited from sleeping to occurring states) by the formatter. This attribute may contain the “indefinite” value, leading to an endless loop of the event occurrences until some external interruption.

Transition names for the event state machine shall be in agreement with Table 23.

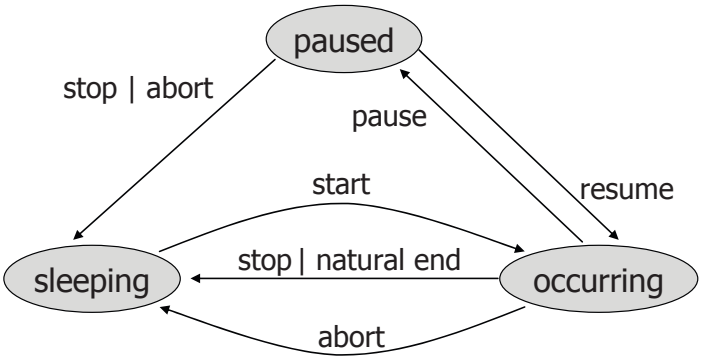

Figure 3 – Event state machine

Table 23 – Transition names for an event state machine

Transition (caused by action)	Transition name
sleeping→ occurring (start)	*starts*
occurring→ sleeping (stop or natural end)	*stops*
occurring→ sleeping (abort)	*aborts*
occurring→ paused (pause)	*pauses*
paused→ occurring (resume)	*resumes*
paused→ sleeping (stop)	*stops*
paused→ sleeping (abort)	*aborts*

© ABNT 2011 - All rights reserved

A presentation event associated with a media node, represented by a <media> element, initializes in the sleeping state. At the beginning of the exhibition of its information units, the event goes to the occurring state. If the exhibition is temporarily suspended, the event stays in the paused state, while this situation lasts.

A presentation event may change from occurring to sleeping as a consequence of the natural end of the presentation duration, or due to an action that stops the event. In both cases, the *occurrences* attribute is incremented, and the *repetitions* attribute is decremented by one. If after being decremented, the *repetitions* attribute value is greater than zero, the event is automatically restarted (set again to the occurring state).

When the presentation of an event is abruptly interrupted, through an abort presentation command, the event also goes to the sleeping state, but without incrementing the *occurrences* attribute and setting the *repetitions* attribute value to zero. The duration of an event is the time it remains in the occurring state. This duration may be intrinsic to the media object, explicitly specified by an author (*explicitDur* attribute of a <descriptor> element), or derived from a relationship.

A presentation event associated with a composite node represented by a <body> or a <context> element stays in the occurring state while at least one presentation event associated with anyone of the composite child nodes is in the occurring state, or at least one context node child link is being evaluated.

A presentation event associated with a composite node represented by a <body: ou <content> element is in the paused state if at least one presentation event associated with anyone of the composite child nodes is in the paused state and all other presentation events associated with the composite child nodes are in the sleeping or paused state. Otherwise, the presentation event is in the sleeping state.

NOTE Others details about the behavior of presentation event state machines for media and composite nodes are given in Clause 8.

A presentation event associated with a switch node, represented by a <switch> element, stays in the occurring state while the switch child element chosen from the bind rules (selected node) is in the occurring state. It is in the paused state if the selected node is in the paused state. Otherwise, the presentation event is in the sleeping state.

A selection event is initiated in the sleeping state. It stays in the occurring state while the corresponding anchor (subset of the information units of a media object) is being selected.

Attribution events stay in the occurring state while the corresponding property values are being modified. Obviously, instantaneous events, like attribution events for simple value assignments, stay in the occurring state only during an infinitesimal period of time.

A composition event (associated to a composite node represented by a <body>, <context> or <switch> element) stays in the occurring state while the composition map is being presented.

Relations are defined based on event states, changes on the event state machines, on event attribute values, and on node (<media>, <body>, <context> or <switch> element) property values. The CausalConnectorFunctionality module allows only the definition of causal relations, defined by the <causalConnector> element of the CausalConnector module.

A <causalConnector> element has a glue expression, which defines a condition expression and an action expression. When the condition expression is satisfied, the action expression shall be executed. The <causalConnector> element shall have the *id* attribute, which uniquely identifies the element within a document.

A condition expression may be simple (<simpleCondition> element) or composite (<compoundCondition> element), both elements defined by the ConnectorCausalExpression module.

The <simpleCondition> element has a *role* attribute, whose value shall be unique in the connector's role set. As aforementioned, a role is a connector interface point, which is associated to node interfaces by a link that refers to the connector. A <simpleCondition> also defines an event type (*eventType* attribute) and to which transition it refers (*transition* attribute). The *eventType* and *transition* attributes are optional. They may be inferred by the *role* value if reserved values are used. Otherwise, the *eventType* and *transition* attributes are required.

© ABNT 2011 - All rights reserved

Reserved values used for defining <simpleCondition> roles are stated in Table 24. If an *eventType* value is “selection”, the role can also define to which selection apparatus (for example, keyboard or remote control keys) it refers, through its *key* attribute. At least the following values (case sensitive) shall be accept for the *key* attribute: “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “A”, “B”, “C”, “D”, “E”, “F”, “G”, “H”, “I”, “J”, “K”, “L”, “M”, “N”, “O”, “P”, “Q”, “R”, “S”, “T”, “U”, “V”, “W”, “X”, “Y”, “Z”, “*”, “#”, “MENU”, “INFO”, “GUIDE”, “CURSOR_DOWN”, “CURSOR_LEFT”, “CURSOR_RIGHT”, “CURSOR_UP”, “CHANNEL_DOWN”, “CHANNEL_UP”, “VOLUME_DOWN”, “VOLUME_UP”, “ENTER”, “RED”, “GREEN”, “YELLOW”, “BLUE”, “BACK”, “EXIT”, “POWER”, “REWIND”, “STOP”, “EJECT”, “PLAY”, “RECORD”, “PAUSE”.

Table 24 – Reserved condition role values associated to event state machines

Role value	Transition value	Event Type
onBegin	*starts*	*presentation*
onEnd	*stops*	*presentation*
onAbort	*aborts*	*presentation*
onPause	*pauses*	*presentation*
onResume	*resumes*	*presentation*
onSelection	*starts*	*selection*
onBeginAttribution	*starts*	*attribution*
onEndAttribution	*stops*	*attribution*

Several reserved words are defined for condition role values. However, it should be noted that there are more values than those defined by the reserved words. Reserved words are only aliases and in the future new aliases can be added. For example, in the case of a selection event, it could be defined reserved words both for the beginning and for the end of a selection.

The role cardinality specifies the minimal (*min* attribute) and maximal (*max* attribute) number of participants that may play the role (number of binds) when the <causalConnector> is used for creating a <link>. The minimal cardinality value shall always be a positive finite value, greater than zero and lesser than or equal to the maximal cardinality value. If minimal and maximal cardinalities are not informed, “1” shall be assumed as the default value for both parameters. When the maximal cardinality value is greater than one, several participants may play the same role, thai is, there may be several binds connecting diverse nodes to the same role. The “unbounded” value may be set to the *max* attribute, if the role may have unlimited binds associated with it. In these two latter cases, a *qualifier* attribute should be specified informing the logical relationship among the simple condition binds. As described in Table 25 the possible values for the *qualifier* attribute are: “or” or “and”. If the qualifier establishes an “or” logical operator, the link action will be fired whenever any condition occurs. If the qualifier establishes an “and” logical operator, the link action will be fired after all the simple conditions occur. If not specified, the default value “or” shall be assumed.

Table 25 – Simple condition *qualifier* values

Role element	Qualifier	Semantics
<simpleCondition>	“or”	True whenever any associated simple condition occurs
<simpleCondition>	“and”	True immediately after all associated simple conditions had occurred

A *delay* attribute may also be defined for a <simpleCondition> specifying that the condition is true after a time delay from the time the transition occurs.

The <compoundCondition> element has a Boolean *operator* attribute (“and” or “or”) relating its child elements: <simpleCondition>, <compoundCondition>, <assessmentStatement> and <compoundStatement>. A *delay* attribute may also be defined specifying that the compound condition is true after a time delay the expression relating its child elements is true. The <assessmentStatement> and <compoundStatement> elements are defined by the ConnectorAssessmentExpression module.

© ABNT 2011 - All rights reserved

NOTE When an “and” compound condition relates more than one trigger condition (that is, a condition that is satisfied only in an infinitesimal time instant – as for example, the end of an object presentation), the compound condition shall be considered true in the instant immediately after all the trigger conditions are satisfied.

An action expression captures actions that may be executed in causal relations and may be composed of a <simpleAction> or a <compoundAction> element, also defined by the ConnectorCausalExpression module.

The <simpleAction> element has a *role* attribute, which shall be unique in the connector role set. As usual, the role is a connector interface point, which is associated to node interfaces by a <link> that refers to the connector. A <simpleAction> also defines an event type (*eventType* attribute) and which event state transition it triggers (*actionType*).

The *eventType* and *actionType* attributes are optional. They can be inferred by the *role* value if reserved values are used. Otherwise, the *eventType* and *actionType* are required. Reserved values used for defining <simpleAction> *roles* are stated in Table 26.

Table 26 – Reserved action role values associated to event state machines

Role value	Action type	Event type
start	*start*	*presentation*
stop	*stop*	*presentation*
abort	*sbort*	*presentation*
pause	*pause*	*presentation*
resume	*resume*	*presentation*
set	*start*	*attribution*

If an *eventType* value is “attribution”, the <simpleAction> shall also define the value that shall be assigned, through its *value* attribute. If the *value* is specified as “$anyName” (where $ is a reserved symbol and anyName is any string, except reserved role names), the assigned value shall be retrieved from the property associated with the *role*=“anyName” and defined by a <bind> child element of the <link> element that refers the connector. If this value cannot be retrieved, no attribution shall be made.

NOTE 1 Declaring the *role*=“anyName” attribute in a <bind> element of a <link> implies having a role implicitly declared as attributeAssessment role=“anyName” eventType=“attribution” attributeType=“nodeProperty”/>. This is the only possible case of a <bind> element refering to a role that is not explicitly declared in a connector.

NOTE 2 In the case that value=“$anyName”, the value to be attributed shall be the value of a property (<property> element) of a component of the same composition where the link (<link> element) that refers to the event is defined, or a property of the composition where the link is defined, or a property of an element that can be reached through a <port> element of the composition where the link is defined, or even a property of an element that can be reached through a port (elements <port> or <switchPort>) of a composition nested in the same composition where the link is defined.

As with <simpleCondition> elements the role cardinality specifies the minimal (*min* attribute) and maximal (*max* attribute) number of participants that may play the role (number of binds) when the <causalConnector> is used for creating a link. When the maximal cardinality value is greater than one, several participants may play the same role. When it has the “unbounded” value, the number of binds is unlimited. In these two later cases, a qualifier shall be specified. Table 27 presents possible qualifier values.

Table 27 – Action qualifier values

Role element	Qualifier	Semantics
simpleAction	“par”	All actions shall be executed in parallel
simpleAction	“seq”	All actions shall be executed in the bind sequence

© ABNT 2011 - All rights reserved

A *delay* attribute may also be defined for a <simpleAction> specifying that the action shall be fired only after waiting for the specified time. Besides, the <simpleAction> may also define a *repeat* attribute to be assigned to the *repetitions* attribute of the event, and a *repeatDelay* to be waited before repeating the action.

Besides all aforementioned attributes, the <simpleAction> element may also have attributes defined in the Animation functionality (*duration* and *by* attributes), if its *eventType* value is “attribution” (see 7.2.13).

The <compoundAction> element has an *operator* attribute (“par” or “seq”) relating its child elements: <simpleAction> and <compoundAction>. Parallel (“par”) and sequential (“seq”) compound actions specify that the execution of actions shall be performed in any order or in a specific order, respectively. A *delay* attribute may also be defined specifying that the compound action shall be applied after the specified delay.

When the sequential operator is used, actions shall be fired in the specified order. However, an action does not need to wait the previous one to be finished in order to be fired.

The ConnectorAssessmentExpression module defines four elements: <assessmentStatement>, <attributeAssessment>, <valueAssessment> and <compoundStatement>.

The <attributeAssessment> has a *role* attribute, which has to be unique in the connector role set. As usual, the *role* is a connector interface point, which is associated to node interfaces by a <link> that refers to the connector. An <attributeAssessment> also defines an event type (*eventType* attribute).

If the *eventType* value is “selection”, the <attributeAssessment> should also define to which selection apparatus (for example, keyboard or remote control keys) it refers, through its *key* attribute. If the *eventType* value is “presentation”, the *attributeType* attribute specifies the event attribute (“occurrences” or “repetition”) or the event state (“state”); if the *eventType* value is “selection”, the *attributeType* attribute is optional and, if present, it may have the value “occurrences” (default) or “state”; if the *eventType* is “attribution” the *attributeType* is optional and may have the value “nodeProperty” (default), “occurrences”, “repetition” or “state”. In the first case, the event represents a node property to be evaluated, in the other ones the event represents the evaluation of the corresponding attribution event property or the attribution event state. An *offset* value may be added to an <attributeAssessment> before the comparison (for example, an offset may be added to an attribute assessment to specify: “the screen vertical position plus 50 pixels”).

The <valueAssessment> element has a *value* attribute that may assume an event state value, or any value to be compared with a node property or event attribute.

The <assessmentStatement> element has a *comparator* attribute that compares the values inferred from its child elements (<attributeAssessment> element and <valueAssessment> element):

a) in the case of <attributeAssessment>: a node property value [*eventType* = “attribution” and the *attributeType* = “nodeProperty”]; or an event attribute value [*eventType* = (“presentation”, “attribution” or “selection”) and the *attributeType* = (“occurrences”, or “repetition”)]; or an event state [*eventType* = (“presentation”, “attribution” or “selection”) and the *attributeType* = “state”];

b) in the case of <valueAssessment>: a value of its *value* attribute.

The <compoundStatement> element has a Boolean *operator* attribute (“and” or “or”) relating its child elements: <assessmentStatement> or <compoundStatement>. An *isNegated* attribute may also be defined to specify that the <compoundStatement> child element shall be negated before the Boolean operation is evaluated.

The <causalConnector> element may have <connectorParam> child elements, which are used to parameterize connector attribute values. The ConnectorCommonPart module defines the type of the <connectorParam> element, which has *name* and *type* attributes.

In order to specify which attributes receive parameter values defined by the connector, their values are specified as the parameter name preceded by the *$* symbol.

EXAMPLE In order to parameterize the *delay* attribute, a parameter called *actionDelay* is defined (<*connectorParam name=”actionDelay” type=”unsignedLong”/*>) and the value *“$actionDelay”* is used in the attribute (*delay*=”$*actionDelay*”).

© ABNT 2011 - All rights reserved

The elements of the CausalConnectorFunctionality module, their attributes and their child elements shall be in agreement with Table 28.

Table 28 – Extended CausalConnectorFunctionality module

Elements	Attributes	Content
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))
connectorParam	*name, type*	empty
simpleCondition	*role, delay, eventType, key, transition, min, max, qualifier*	empty
compoundCondition	*operator, delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay, duration, by*	empty
compoundAction	*operator, delay*	(simpleAction | compoundAction)+
assessmentStatement	*comparator*	(attributeAssessment, (attributeAssessment | valueAssessment))
attributeAssessment	*role, eventType, key, attributeType, offset*	empty
valueAssessment	*value*	empty
compoundStatement	*operator, isNegated*	(assessmentStatement | compoundStatement)+

The ConnectorBase module defines an element named <connectorBase>, which allows grouping connectors. As usual, the <connectorBase> element should have the *id* attribute, which uniquely identifies the element within a document.

The exact content of a connector base is specified by the language profile that uses the connectors facility. However, since the definition of connectors is not easily done by naïve users, the idea is to have expert users defining connectors, storing them in libraries (connector bases) that may be imported, and making them available to others for creating links. Annex C gives an extensive example of connector definitions that may be imported.

The element of the ConnectorBase module, its attributes, and its child elements shall be in agreement with Table 29.

© ABNT 2011 - All rights reserved

Table 29 – Extended ConnectorBase module

Elements	Attributes	Content
connectorBase	*id*	(importBase|causalConnector)*

7.2.9 Presentation control functionality

The purpose of the Presentation Control functionality is to specify content and presentation alternatives for a document. This functional area is partitioned into four modules, named TestRule, TestRuleUse, ContentControl and DescriptorControl.

The TestRule module allows the definition of rules that, when satisfied, select alternatives for document presentation. The specification of rules in NCL 3.0 is done in a separate module, because they are useful for defining either alternative components or alternative descriptors.

The <ruleBase> element specifies a set of rules, and shall be defined as a child element of the <head> element. These rules may be simple, defined by the <rule> element, or composite, defined by the <compositeRule> element. Simple rules define an identifier (*id* attribute), a variable (*var* attribute), a value (*value* attribute), and a comparator (*comparator* attribute) relating the variable to the value.

The variable shall be a property of the settings node (<media> element of application/x-ginga-settings type), that is, the *var* attribute shall have the same value of a <property> *name* attribute, defined as a child of the <media> element of application/x-ginga-settings type. Composite rules have an identifier (*id* attribute) and a Boolean operator ("and" or "or" – *operator* attribute) relating their child rules. As usual, the *id* attribute uniquely identifies the <rule> and <compositeRule> elements within a document.

The elements of the TestRule module, their attributes, and their child elements shall be in agreement with Table 30.

Table 30 – Extended TestRule module

Elements	Attributes	Content
Rule	*id, var, comparator, value*	empty
rule	*id, var, comparator, value*	empty
compositeRule	*id, operator*	(rule | compositeRule)+

The TestRuleUse defines the <bindRule> element, which is used to associate rules with components of a <switch> or <descriptorSwitch> element, through its *rule* and *constituent* attributes, respectively.

The element of the TestRuleUse module and its attributes shall be in agreement with Table 31.

Table 31 – Extended TestRuleUse module

Elements	Attributes	Content
bindRule	*constituent, rule*	empty

© ABNT 2011 - All rights reserved

The ContentControl module specifies the <switch> element, allowing the definition of alternative document nodes to be chosen during presentation time. Test rules used to choose the switch component to be presented are defined by the TestRule module or are test rules specifically defined and embedded in an NCL formatter implementation. The ContentControl module also defines the <defaultComponent> element, whose *component* attribute (also of IDREF type) identifies the default element that shall be selected if none of the bindRule rules is evaluated as true.

In order to allow links to anchor on the component chosen after evaluating the rules of a *switch*, a language profile should also include the SwitchInterface module, which allows the definition of special interfaces, named <switchPort>.

As usual, <switch> elements shall have the *id* attribute, which uniquely identifies the element within a document. The *refer* attribute is an extension defined in the Reuse module (see 7.2.11).

When a <context> is defined as a child of a <switch> element, the <link> elements recursively contained in the <context> shall be considered by an NCL player only if the <context> is selected after the switch evaluation. Otherwise, the <link> elements should be considered disabled and shall not interfere in the document presentation.

The ContentControl module elements, their attributes and their child elements shall be in agreement with a Table 32.

Table 32 – Extended ContentControl module

Elements	Attributes	Content
switch	*id, refer*	defaultComponent?, (switchPort | bindRule | media | context | switch)*)
defaultComponent	*component*	empty

The DescriptorControl module specifies the <descriptorSwitch> element, which contains a set of alternative descriptors to be associated with an object. The <descriptorSwitch> elements shall have the *id* attribute, which uniquely identifies the element within a document. Analogous to the <switch> element, the <descriptorSwitch> choice is done during presentation time, using test rules defined by the TestRule module, or test rules specifically defined and embedded in an NCL formatter implementation. The DescriptorControl module also defines the <defaultDescriptor> element, whose *descriptor* attribute (also of IDREF type) identifies the default element that shall be selected if none of the bindRule rules is evaluated as true.

The DescriptorControl module elements, their attributes, and their child elements shall be in agreement with Table 33.

Table 33 – Extended DescriptorControl module

Elements	Attributes	Content
descriptorSwitch	*id*	(defaultDescriptor?, (bindRule | descriptor)*)
defaultDescriptor	*descriptor*	empty

During a document presentation, from the moment on a <switch> is evaluated, it is considered resolved until the end of the current switch presentation, that is, while its corresponding presentation event is in the “occurring” or “paused” state. During a document presentation, from the moment on a <descriptorSwitch> is evaluated, it is considered resolved until the end of the presentation of the <media> element that was associated to it, that is, while any presentation event associated with the <media> element is in the “occurring” or “paused” state.

© ABNT 2011 - All rights reserved

NOTE NCL formatters should delay the switch evaluation to the moment that a link anchoring in the switch needs to be evaluated. The descriptorSwitch evaluation should be delayed until the object referring the descriptorSwitch needs to be prepared to be presented.

7.2.10 Timing functionality

The Timing functionality defines the Timing module. The Timing module allows the definition of temporal attributes for document components. Basically, this module defines attributes for specifying what will happen with an object at the end of its presentation (*freeze*), and the ideal duration of an object (*explicitDur*). These attributes may be incorporated by <descriptor> elements.

7.2.11 Reuse functionality

NCL allows intensive reuse of its elements. The NCL Reuse functionality is partitioned into three modules: Import, EntityReuse and ExtendedEntityReuse.

In order to allow an entity base to incorporate another already-defined base, the Import module defines the <importBase> element, which has two attributes: *documentURI* and *alias*. The *documentURI* refers to a URI corresponding to the NCL document containing the base to be imported. The *alias* attribute specifies a name to be used as prefix when referring to elements of this imported base.

The alias name shall be unique in a document and its scope is constrained to the document that has defined the *alias* attribute. The reference would have the format: *alias#element_id*. The import operation is transitive, that is, if *baseA* imports *baseB* that imports *baseC*, then *baseA* imports *baseC*. However, the *alias* defined for *baseC* inside *baseB* shall not be considered by *baseA*.

When a language profile uses the Import module, the following specifications are allowed:

— the <descriptorBase> element may have a child <importBase> element referring to a URI corresponding to another NCL document containing the descriptor base (in fact its child elements) to be imported and nested. When a descriptor base is imported, the region base and the rule base, when present in the imported document, are also automatically imported to the corresponding region and rule bases of the importing document;

— the <connectorBase> element may have a child <importBase> element referring to a URI corresponding to another connector base (in fact its child elements) to be imported and nested;

— the <transitionBase> element may have a child <importBase> element referring to a URI corresponding to another transition base (in fact its child elements) to be imported and nested;

— the <ruleBase> element may have a child <importBase> element referring to a URI corresponding to another NCL document containing the rule base (in fact its child elements) to be imported and nested;

— the <regionBase> element may have a child <importBase> element referring to a URI corresponding to another NCL document containing the region base (in fact its child elements) to be imported and nested. As the referred document URI can have more than one region base, the base to be imported must be identified by assigning its *id* to the *baseId* attribute. Although NCL defines its layout model, nothing prevents an NCL document from using other layout models, since they define regions where objects may be presented, as for example SMIL 2.1 layout models. On importing a <regionBase>, an optional attribute named *region* may be specified within the <importBase> element. When present, the attribute shall identify the *id* of a <region> element declared in the <regionBase> element of the host document (the document that did the importing operation). As a consequence, all child <region> elements of the imported <regionBase> shall be considered as child <region> elements of the region referred by the <importBase>'s *region* attribute. If not specified, the child <region> elements of the imported <regionBase> shall be considered children of the host document <regionBase> element.

© ABNT 2011 - All rights reserved

The <importedDocumentBase> element specifies a set of imported NCL documents, and shall be defined as a child element of the <head> element. In addition, <importedDocumentBase> elements may have the *id* attribute, which uniquely identifies the element within a document.

An NCL document may be imported through the <importNCL> element. All bases defined inside an NCL document, as well as the document <body> element, are imported all at once through the <importNCL> element. The bases are treated as if each one is imported by an <importBase> element. The imported <body> element will be treated as a <context> element. It should be stressed that the <importNCL> element does not "include" the referred NCL document but only makes the referred document visible to have its components reused by the document that has defined the <importNCL> element. Thus, imported <body>, as well as any of its contained nodes, may be reused inside the <body> element of the importing NCL document.

The <importNCL> element has two attributes: *documentURI*, and *alias*. The *documentURI* refers to a URI corresponding to the document to be imported. The *alias* attribute specifies a name to be used when referring an element of this imported document. As in the <importBase> element, the name shall be unique (type=ID) and its scope is constrained to the document that has defined the *alias* attribute. The reference would have the format: *alias#element_id*. It is important to note that the same alias should be used when referring to elements defined in the imported document bases (<regionBase>, <connectorBase>, <descriptorBase>, etc.).

The <importNCL> element operation has also the transitive property, that is, if *documentA* imports *documentB* that imports *documentC*, then *documentA* imports *documentC*. However, the *alias* defined for *documentC* inside *documentB* shall not be considered by *documentA*.

The elements of the Import module, their child elements, and their attributes shall be in agreement with Table 34.

Table 34 – Extended Import module

Elements	Attributes	Content
importBase	*alias, documentURI, region, baseId*	empty
importedDocumentBase	*id*	(importNCL)+
importNCL	*alias, documentURI*	empty

The EntityReuse module allows an NCL element to be reused. This module defines the *refer* attribute, which refers to an element *id* that will be reused. Only <media>, <context>, <body> and <switch> may be reused. An element that refers to another element cannot be reused; that is, its *id* cannot be the value of any *refer* attribute.

If the referred node is defined within an imported document *D*, the *refer* attribute value shall have the format "alias#id", where "alias" is the value of the *alias* attribute associated with the *D* import.

When a language profile uses this module, it may add the *refer* attribute to:

— a <media> or <switch> element. In this case, the referred element shall be, respectively, a <media> or <switch> element, which represents the same node previously defined in the document <body> itself or in an external imported <body>. This referred element shall directly contain the definition of all its attributes and child elements;

— a <context> element. In this case, the referred element shall be a <context> or a <body> element that will represent the same context, which is previously defined in the document <body> itself or in an external imported <body>. This referred element shall directly contain the definition of all its attributes and child elements.

© ABNT 2011 - All rights reserved

When an element declares a *refer* attribute, all attributes and child elements defined by the referred element are inherited. All other attributes and child elements, if they are defined by the referring element, shall be ignored by the formatter, except the *id* attribute that shall be defined. The only other exception is for <media> elements, in which new child <area> and <property> elements may be added, and a new attribute, *instance*, may be defined.

If the new added <property> element has the same *name* attribute of an already existing <property> element (defined in the reused <media> element), the new added <property> shall be ignored. Similarly, if the new added <area> element has the same *id* attribute of an already existent <area> element (defined in the reused <media> element), the new added <area> shall be ignored. The *instance* attribute is defined in the ExtendedEntityReuse module and has “new” as its default string value.

The referred element and the element that refers to it shall be considered the same, regarding its data specification. In other words it means that a single NCM node can be represented by more than one NCL element. As nodes contained in an NCM composite node define a set, an NCM node may be represented by no more than one NCL element inside a composition. This means that the *id* attribute of an NCL element representing an NCM node is not only a unique identifier for the element, but also the unique identifier for the NCM node in the composition.

NOTE Other information can be found in NCMCore:2005.

EXAMPLE Assume the NCL element (*node1*) that defines an NCM node. The NCL elements that refer to it (*node1ReuseA, node1ReuseB*) represent the same NCM node. In other words, the single NCM node is represented by more than one NCL element (*node1, node1ReuseA,* and *node1ReuseB*). Moreover, since nodes contained in an NCM composite node define a set, the NCL elements *node1, node1ReuseA,* and *node1ReuseB* shall each be declared inside a different composition.

The referred element and the element that refers to it shall also be considered the same regarding their presentation, if the *instance* attribute receives a “instSame” or “gradSame” value. Therefore, the following semantics shall be respected. Assume the set of <media> elements composed of the referred <media> element and all the referring <media> elements. If any element of the subset formed by the referred <media> element and all other <media> elements having the *instance* attribute equal to “instSame” or “gradSame” is scheduled to be presented, all other elements in this subset, which are not child descendents of a <switch> element, are also assumed as scheduled for presenting, and more than that, when they are being presented, they shall be represented by the same presentation instance. Descendent elements of a <switch> element shall also have the same behavior, if all rules needed to present these elements are satisfied; otherwise they shall not be scheduled for presenting. If the *instance* attribute is equal to “instSame”, all scheduled nodes of the subset shall be immediately presented through a unique instance (start instruction applied on all subset elements). If the *instance* attribute is equal to “gradSame”, all scheduled nodes of the subset shall be presented through a unique instance, but now gradually, as while start instructions are applied, coming from a link, etc. The common instance in presentation shall notify all events associated with the <area> and <property> elements defined in all <media> elements of this subset that were scheduled for presenting. On the other hand, the <media> elements in the set that have *instance* attribute values equal to “new” shall not be scheduled for presenting. When they are individually scheduled for presenting, no other element in the set is affected. Moreover, new independent presentation instances shall be created at each individual presentation starting.

7.2.12 Navigational Key Functionality

The Navigational Key functionality defines the KeyNavigation module that 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 *focusIndex* attribute specifies an index for the <media> element to which the focus may be applied, when this element is in exhibition. The *focusIndex* may be defined using a <property> or a <descriptor> element. When this property is not defined, the object is considered as if no focus could be set. In a certain presentation moment, if the focus has not been already defined, or is lost, a focus will be initially applied to the element that is being presented with the smallest index value.

Values of *focusIndex* attribute shall be unique in an NCL document. Otherwise, the repeated attributes will be ignored if in a certain moment there is more than one <media> element to gain the focus. Moreover, when a <media> element refers to another <media> element (using the *refer* attribute specified in Section 7.2.11), it shall ignore the *focusIndex* associated with the referred <media> element.

© ABNT 2011 - All rights reserved

The *moveUp* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus should be applied when the “up arrow key” is pressed. The *moveDown* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus should be applied when the “down arrow key” is pressed.

The *moveRight* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus should be applied when the “right arrow key” is pressed. The *moveLeft* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus should be applied when the “left arrow key” is pressed.

When the focus is applied to an element with the visible property set to false, or to an element that it is not being presented, the current focus does not move.

The *focusSrc* attribute can specify an alternative media source to be presented, instead of the current presentation, if an element receives the focus. This attribute follows the same rules of the *src* attribute of the <media> element.

When an element receives a focus, the square box defined by the element positioning attributes shall be decorated. The *focusBorderColor* attribute defines the decorative color and may receive the reserved color names: “white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal”. The *focusBorderWidth* attribute defines the width in pixels of the decorative border (0 means that no border will appear, positive values means that the border is outside the object content, and negative values means that the border is drawn over the object content), and the *focusBorderTransparency* attribute defines the decorative color transparency. The *focusBorderTransparency* shall be a real value between 0 and 1, or a real value in the range [0,100] ending with the character “%” (for example, 30 %), with “1” or “100 %” meaning full transparency and “0” or “0 %” meaning no transparency. When the *focusBorder Color,* the *focusBorderWidth,* or the *focusBorderTransparency* are not defined, default values shall be assumed. These values are specified in properties of the <media> element of application/x-ginga-settings type: *default.focusBorderColor, default.focusBorderWidth, default.focusTransparency*, respectively.

When an element on focus is selected by pressing the activation (select or enter) key, the *focusSelSrc* attribute can specify an alternative media source to be presented, instead of the current presentation. This attribute follows the same rules of the *src* attribute of the <media> element. When selected, the square box defined by the element positioning attributes shall be decorated with the color defined by the *selBorderColor* attribute (default value specified by the *default.selBorderColor* of the <media> element of application/x-ginga-settings type), the width of the decorative border defined by the *focusBorderWidth* attribute, and the decorative color transparency defined by the *focusBorderTransparency* attribute.

When an element on focus is selected by pressing the “activate (select or enter) key”, the focus control shall be passed to the <media> element renderer (player). The player can then follow its own rules for navigation. The focus control shall be passed back to the NCL formatter when the “back key” is pressed. In this case, the focus goes to the element identified by the *service.currentFocus* atribute of the *settings* node (<media> element of application/x-ginga-settings type).

The focus control may also be passed by setting the service.currentKeyMaster attribute of the *settings* node (<media> element of application/x-ginga-settings type). This may be done through a link action, through an NCL editing command executed by an imperative-code node (NCLua or NCLet object, for example). The player of a node that has the current control may not directly change the service.currentKeyMaster property.

7.2.13 Animation functionality

Animation in the cartoon sense is actually a combination of two factors: support for object drawing and support for object motion or more correctly, support for object alteration as a function of time.

NCL is not a content format and, as such, does not have support for creating media objects and it does not have a generalized method for altering media object content. Instead, NCL is a scheduling and orchestration format. This means that NCL cannot be used to make cartoons, but can be used to render cartoon objects in the context of a general presentation, and to change the timing and rendering properties of a cartoon (or any other) object as a whole, while it is being displayed.

© ABNT 2011 - All rights reserved

The animation primitives of NCL allow values of node properties to be changed during an active explicitly declared duration. Since NCL animation can be computationally intensive, it is only supported by the EDTV profile and only the properties that define numerical values and colors may be animated.

The Animation Functionality defines the Animation module that provides the extensions necessary to describe what happens when a node property value is changed. Basically, the module defines attributes that may be incorporated by <simpleAction> elements of a connector, if its *eventType* value is “attribution”. Two new attributes are defined: *duration* and *by*.

When setting a new value to a property the change is instantaneous by default (*duration*=″0″), but the change may also be carried out during an explicitly declared duration, specified by the *duration* attribute.

Also, when setting a new value to a property the change from the old value to the new one may be linear by default (*by*=″indefinite″), or carried out step by step, with the pace specified by the *by* attribute.

The combination of the *duration* and *by* attribute definitions gives how (discretely or linearly) the change shall be performed and its transforming interval.

7.2.14 Transition Effects functionality

The Transition Effects functionality is divided into two modules: TransitionBase and Transition.

NOTE Other information can be found in SMIL 2.1 Specification:2005.

The TransitionBase module is defined by NCL 3.0 and consists on the <transitionBase> element that specifies a set of transition effects, and shall be defined as a child element of the <head> element.

The <transitionBase> element, its child elements, and its attributes shall be in agreement with Table 35.

Table 35 – Extended TransitionBase module

Elements	Attributes	Content
transitionBase	*id*	(importBase, transition)+

The Transition module is based on SMIL 2.1 specifications [SMIL 2.1 Specification, 2005]. It has just one element called <transition>.

In NCL 3.0 Enhanced DTV profile, the <transition> element is specified in the <transitionBase> element and allows a transition template to be defined. Each <transition> element defines a single transition template and shall have an *id* attribute so that it may be referred.

Seven <transition> element’s attributes come from SMIL BasicTransitions module specification: *type*; *subtype*; *dur*; *startProgress*; *endProgress*; *direction*; and *fadeColor*.

Transitions are classified according to a two-level taxonomy of types and subtypes. Each of the transition types describe a group of transitions which are closely related. Within that type, each of the individual transitions is assigned a subtype which emphasizes the distinguishing characteristic of that transition.

The *type* attribute is required and is used to specify the general transition. If the named type is not supported by the NCL formatter, the transition is ignored. This is not an error condition, since implementations are free to ignore transitions.

The *subtype* attribute provides transition-specific control. This attribute is optional and, if specified, shall be one of the transition subtypes appropriate for the specified type. If this attribute is not specified then the transition reverts to the default subtype for the specified transition type. Only the subtypes for the five required transition types listed in Table 36 shall be supported, the others, defined in SMIL specifications are optional.

© ABNT 2011 - All rights reserved

Table 36 – Required transition types and subtypes

Transition type	Default transition subtype
barWipe	leftToRight
irisWipe	rectangle
clockWipe	clockwiseTwelve
snakeWipe	topLeftHorizontal
fade	crossfade

The *dur* attribute specifies the duration of the transition. The default duration is 1 s.

The *startProgress* attribute specifies the amount of progress through the transition at which to begin execution. Legal values are real numbers in the range [0.0,1.0]. For instance, we can want to begin a crossfade with the destination image being already 30 % faded in. For this case, *startProgress* would be 0.3. The default value is 0.0.

The *endProgress* attribute specifies the amount of progress through the transition at which to end execution. Legal values are real numbers in the range [0.0,1.0], and the value of this attribute shall be greater than or equal to the value of the *startProgress* attribute. If *endProgress* is equal to *startProgress*, then the transition remains at a fixed progress for the duration of the transition. The default value is 1.0.

The *direction* attribute specifies the direction the transition will run. The legal values are “forward” and “reverse”. The default value is “forward”. Not all transitions will have meaningful reverse interpretations. For instance, a crossfade is not a geometric transition, and therefore has no interpretation of reverse direction. Transitions that do not have a reverse interpretation should have the *direction* attribute ignored and the default value of “forward” assumed.

If the value of the *type* attribute is “fade” and the value of the *subtype* attribute is “fadeToColor” or “fadeFromColor” (values that are not required to be supported in a Ginga implementation), then the *fadeColor* attribute specifies the ending or starting color of the fade. If the value of the *type* attribute is not “fade”, or the value of the *subtype* attribute is not “fadeToColor” or “fadeFromColor”, then the *fadeColor* attribute shall be ignored. The default value is “black”.

The Transition module also defines attributes to be used in <descriptor> elements to use the transition templates defined by <transition> elements: *transIn* and *transOut* attributes. Transitions specified with a *transIn* attribute will begin at the beginning of the media element's active duration (when the object presentation begins to occur). Transitions specified with a *transOut* attribute will end at the end of the media element's active duration (when the object presentation transits from occurring to sleeping state).

The *transIn* and *transOut* attributes are added to <descriptor> elements. The default value of both attributes is an empty string, which indicates that no transition shall be performed. The properties may also be defined using <property> elements.

The value of the *transIn* and *transOut* attributes is a semicolon-separated list of transition identifiers. Each of the identifiers shall correspond to the value of the XML identifier of one of the transition elements previously defined in the <transitionBase> element. The purpose of the semicolon-separated list is to allow authors to specify a set of fallback transitions if the preferred transition is not available.

The first transition in the list should be performed if the user-agent has implemented this transition. If this transition is not available, then the second transition in the list should be performed, and so on. If the value of the *transIn* or *transOut* attribute does not correspond to the value of the XML identifier of any one of the transition elements

© ABNT 2011 - All rights reserved

previously defined, then this is an error. In the case of this error, the value of the attribute should be considered to be the empty string and therefore no transition should be performed.

All transitions defined in the Transition module accept four additional attributes (coming from the SMIL TransitionModifiers module specification) that may be used to control the visual appearance of the transitions. The *horzRepeat* atribute specifies how many times to perform the transition pattern along the horizontal axis. The default value is 1 (the pattern occurs once horizontally). The *vertRepeat* attribute specifies how many times to perform the transition pattern along the vertical axis. The default value is 1 (the pattern occurs once vertically). The *borderWidth* attribute specifies the width of a generated border along a wipe edge. Legal values are integers greater than or equal to 0. If *borderWidth* value is equal to 0, then no border should be generated along the wipe edge. The default value is 0. If the value of the *type* attribute is not “fade”, then the *borderColor* attribute specifies the content of the generated border along a wipe edge. If the value of this attribute is a color, then the generated border along the wipe or warp edge is filled with this color. If the value of this attribute is "blend", then the generated border along the wipe blend is an additive blend (or blur) of the media sources. The default value for this attribute is "black".

The element of the Extended Transition Module, its child elements, and its attributes shall be in agreement with Table 37.

Table 37 – Extended Transition module

Elements	Attributes	Content
Transition	*id, type, subtype, dur,startProgress, endProgress, direction, fadeColor, horzRepeat, vertRepeat, borderWidth, borderColor*	*empty*

7.2.15 Metainformation functionality

Metainformation does not contain content information that is used or displayed during a presentation. Instead, it contains information about content that is used or displayed. The Metainformation Functionality is composed of the Metainformation module that comes from SMIL Metainformation module specification.

NOTE Other information can be found in SMIL 2.1 Specification:2005.

The Metainformation module contains two elements that allow description of NCL documents. The <meta> element specifies a single property/value pair in the *name* and *content* attributes, respectively. The <metadata> element contains information that is also related to metainformation of the document. It acts as the root element of the RDF tree. The <metadata> element may have as child elements: RDF elements and its sub-elements [RDF, 1999].

NOTE Other information can be found in RDF:1999.

The elements of the Metainformation module, their child elements, and their attributes shall be in agreement with Table 38.

Table 38 – Extended Metainformation module

Elements	Attributes	Content
meta	*name, content*	empty
metadata	*empty*	RDF tree

© ABNT 2011 - All rights reserved

7.3 NCL language profiles for SBTVD

7.3.1 Profiles modules

Each NCL profile may group a subset of NCL modules, allowing the creation of languages according to user needs.

Any document in conformance with NCL profiles shall have the <ncl> element as its root element.

The NCL 3.0 Full profile, also called NCL 3.0 Language profile, is the “complete profile” of the NCL 3.0 language. It comprises all NCL modules (including those discussed in 7.2) and provides all facilities for declarative authoring of NCL documents.

The profiles defined for the SBTVD are:

a) NCL 3.0 Enhanced DTV profile: includes the Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, PropertyAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse KeyNavigation, Animation, TransitionBase, Transition and Metainformation modules of NCL 3.0. The tables of 7.2 show each module element, already extended by the attributes and child elements inherited from other modules, for this profile.(see XML schemas in 7.3.2).;

b) NCL 3.0 CausalConnector profile: allows the creation of simple hypermedia connectors. This profile includes the Structure, CausalConnectorFunctionality, and ConnectorBase modules. In the profile, the <body> element of the Structure module is not used (see XML schemas in 7.3.3);

c) NCL 3.0 Basic DTV profile: includes the Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, PropertyAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse and KeyNavigation modules. The tables of 7.3.4 show each module element for this profile, already extended by the attributes and child elements inherited from other modules. (see XML schemas in 7.3.5).

© ABNT 2011 - All rights reserved

7.3.2 The Schema of the NCL 3.0 Enhanced DTV Profile

NCL30EDTV.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/NCL30EDTV.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:metainformation="http://www.w3.org/2001/SMIL20/Metainformation"
  xmlns:basicTransition="http://www.w3.org/2001/SMIL20/BasicTransitions"
  xmlns:profile="http://www.ncl.org.br/NCL3.0/EDTVProfile"
  targetNamespace="http://www.ncl.org.br/NCL3.0/EDTVProfile"
  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"/>
```

© ABNT 2011 - All rights reserved

```
<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"/>


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


© ABNT 2011 - All rights reserved

```
<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:importBase"/>
        <element ref="profile:region"/>
      </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">
```

© ABNT 2011 - All rights reserved

```
      <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">
    </extension>
  </complexContent>
</complexType>

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

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

<complexType name="compositeNodePortType">
  <complexContent>
```


© ABNT 2011 - All rights reserved

```
      <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">
```

© ABNT 2011 - All rights reserved

```
        <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"/>
```


© ABNT 2011 - All rights reserved

```
  <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>
    </complexContent>
  </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">
```

© ABNT 2011 - All rights reserved

```
   <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">
```


© ABNT 2011 - All rights reserved

```
    <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 -->
```

© ABNT 2011 - All rights reserved

```
  <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>
```

7.3.3 The schema of the NCL 3.0 CausalConnector profile

CausalConnector.xsd

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

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

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

-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
        xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
        xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
        xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
        xmlns:profile="http://www.ncl.org.br/NCL3.0/CausalConnectorProfile"
        targetNamespace="http://www.ncl.org.br/NCL3.0/CausalConnectorProfile"
        elementFormDefault="qualified" attributeFormDefault="unqualified">

    <!-- import the definitions in the modules namespaces -->

    <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/Structure"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Structure.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/Import"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Import.xsd"/>
```


© ABNT 2011 - All rights reserved

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

  <complexType name="nclType">
    <complexContent>
      <restriction base="structure:nclPrototype">
        <sequence>
          <element ref="structure:head" minOccurs="0" maxOccurs="1"/>
          <element ref="structure:body" minOccurs="0" maxOccurs="0"/>
        </sequence>
      </restriction>
    </complexContent>
  </complexType>

  <element name="ncl" type="profile:nclType" substitutionGroup="structure:ncl"/>

  <!-- extends head element -->

  <complexType name="headType">
    <complexContent>
      <extension base="structure:headPrototype">
        <all>
          <element ref="profile:connectorBase" />
        </all>
      </extension>
    </complexContent>
  </complexType>

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

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


<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" substitutionGroup="causalConnectorFunctionality:simpleAction"/>

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

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

© ABNT 2011 - All rights reserved

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

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

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

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

  <element name="importBase" substitutionGroup="import:importBase"/>

</schema>
```

7.3.4 Attributes and elements of the NCL 3.0 Basic DTV profile

The elements and the attributes used in NCL 3.0 Basic DTV profile are shown in Tables 39 to 55. Note that attributes and contents (child elements) of elements may be defined in the module itself or in the NCL Basic DTV profile that groups the modules. Element attributes that are required are underlined. In the Tables 39 to 55, the following symbols are used: (?) optional (zero or one occurrence), (|) or, (*) zero or more occurrences, (+) one or more occurrences.

Table 39 – Extended structure module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
Ncl	*id, title, xmlns*	(head?, body?)
Head		(importedDocumentBase? ruleBase?, regionBase*, descriptorBase?, connectorBase?),
Body	*id*	(port| property| media|context|switch|link)*

Table 40 - Extended layout module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
regionBase	*id, device, region*	(importBase|region)+
Region	*id, title, left, right, top, bottom, height, width, zIndex*	(region)*

© ABNT 2011 - All rights reserved

Table 41 – Extended media module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
media	*id, src, refer, instance, type, descriptor*	(area|property)*

Table 42 – Extended context module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
context	*id, refer*	(port|property|media|context|link|switch)*

Table 43 – Extended MediaContentAnchor module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
area	*id, coords, begin, end, beginText, beginPosition endText, endPosition, first, last, label, clip*	empty

Table 44 – Extended CompositeNodeInterface module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
port	*id, component, interface*	empty

Table 45 – Extended PropertyAnchor module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
property	*name, value*	empty

Table 46 – Extended SwitchInterface module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
switchPort	*id*	mapping+
mapping	*component, interface*	empty

© ABNT 2011 - All rights reserved

Table 47 – Extended descriptor module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
descriptor	*id, player, explicitDur, region, freeze, moveLeft, moveRight, moveUp; moveDown, focusIndex, focusBorderColor; focusBorderWidth; focusBorderTransparency, focusSrc,focusSelSrc, selBorderColor*	(descriptorParam)*
descriptorParam	*name, value*	
descriptorBase	*id*	(importBase | descriptor | descriptorSwitch)+

Table 48 - Extended linking module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
bind	*role, component, interface, descriptor*	(bindParam)*
bindParam	*name, value*	empty
linkParam	*name, value*	empty
link	*id, xconnector*	(linkParam*, bind+)

Table 49 – Extended CausalConnector functionality module elements and attributes in the Basic DTV profile

Elements	Attributes	Content
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))
connectorParam	*name, type*	empty
simpleCondition	*role, delay, eventType, key, transition, min, max, qualifier*	empty
compoundCondition	*operator, delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay*	empty
compoundAction	*operator, delay*	(simpleAction | compoundAction)+
assessmentStatement	*comparator*	(attributeAssessment, (attributeAssessment | valueAssessment))
attributeAssessment	*role, eventType, key, attributeType, offset*	*vazio*
valueAssessment	*value*	empty
compoundStatement	*operator, isNegated*	(assessmentStatement | compoundStatement)+

© ABNT 2011 - All rights reserved

Table 50 – Extended ConnectorBase module element and attributes used in the Basic DTV profile

Elements	Attributes	Content
connectorBase	*id*	(importBase|causalConnector)*

Table 51 – Extended TestRule Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
ruleBase	*id*	(importBase|rule|compositeRule)+
Rule	*id, var, comparator, value*	vazio
compositeRule	*id, operator*	(rule | compositeRule)+

Table 52 – Extended TestRuleUse module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
bindRule	*constituent, rule*	empty

Table 53 – Extended ContentControl module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
switch	*id, refer*	(defaultComponent?,(switchPort| bindRule|media| context | switch)*)
defaultComponent	*component*	empty

Table 54 – Extended DescriptorControl module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
descriptorSwitch	*id*	(defaultDescriptor?, (bindRule | descriptor)*)
defaultDescriptor	*descriptor*	empty

Table 55 – Extended import module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
importBase	*alias, documentURI, region*	empty
importedDocumentBase	*id*	(importNCL)+
importNCL	*alias, documentURI,*	empty

© ABNT 2011 - All rights reserved

7.3.5 The schema of the NCL 3.0 Basic DTV profile

NCL30BDTV.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/NCL30BDTV.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
 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: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:profile="http://www.ncl.org.br/NCL3.0/BDTVProfile"
 targetNamespace="http://www.ncl.org.br/NCL3.0/BDTVProfile"
 elementFormDefault="qualified" attributeFormDefault="unqualified" >

 <!-- import the definitions in the modules namespaces -->
 <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/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"
```


© ABNT 2011 - All rights reserved

```
  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"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- 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:regionBase" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="profile:descriptorBase" minOccurs="0" maxOccurs="1"/>
        <element ref="profile:connectorBase" minOccurs="0" maxOccurs="1"/>
      </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"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

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

© ABNT 2011 - All rights reserved

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


© ABNT 2011 - All rights reserved

```xml
</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"/>
      </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">
    </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 -->
```

© ABNT 2011 - All rights reserved

```
<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"/>
    </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>
```

 © ABNT 2011 - All rights reserved

```
</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>

<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" 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"/>
```

© ABNT 2011 - All rights reserved

```
  <!-- extends compositeRule element -->
  <complexType name="compositeRuleType">
    <complexContent>
      <extension base="testRule:compositeRulePrototype">
      </extension>
    </complexContent>
  </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">
```


© ABNT 2011 - All rights reserved

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

</schema>
```

8 Media objects in NCL presentations

8.1 A modular Ginga-NCL implementation

The presentation of an NCL document requires the synchronization control of several media objects specified through the <media> element. For each media object, a media player can be loaded to control the object and its NCL events. A media player shall be able to receive presentation commands, to control the events' state machines of the controlled media object, and answer queries coming from the formatter.

In order to favor the incorporation of third-party media players into the Ginga architecture implementation, a modular design of Ginga-NCL is recommended, aiming at separating the media players from the presentation engine (NCL formatter).

The Figure 4 suggests a modular organization for the Ginga-NCL implementation. The media players are plug-in modules of the presentation engine. Since it can be interesting to use already existing media players that can have proprietary interfaces that are not compatible with the one required by the presentation engine, it will be necessary to develop modules to make the necessary adaptations. In this case, the media player will be constituted of an adapter besides the player itself.

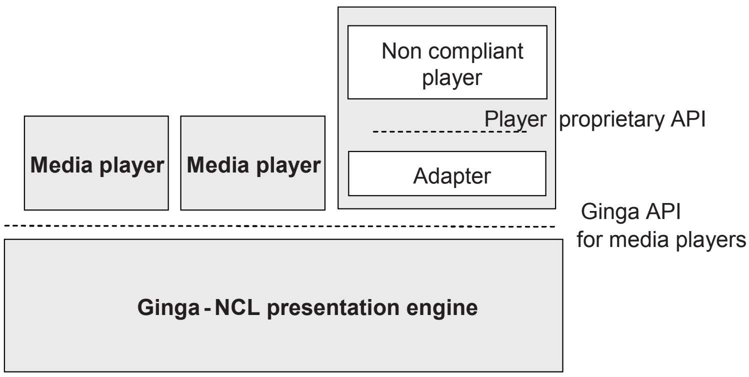

Figure 4 – API for integrating media players with an NCL presentation engine implementation

© ABNT 2011 - All rights reserved

NOTE As the Ginga-NCL architecture and implementation is a choice of each receiver developer, the next sections do not intend to standardize the syntax of the presentation engine API. The goal is just to define the expected behavior of a media player when controlling objects that take part in an NCL document.

8.2 Expected behavior of basic media players

8.2.1 Start instruction for presentation events

Before sending a *start* instruction, the formatter should find the more appropriate media player to be called based on the content type to be exhibited. For this sake, the formatter takes into consideration the *player* attribute associated with the media object to be exhibit. If this attribute is not specified, the formatter shall take into account the *type* attribute of the <media> element. If this attribute is not specified either, the formatter shall consider the file extension specified in the *src* attribute of the <media> element.

The *start* instruction issued by a formatter shall inform the following parameters to the media player: the media object to be controlled, its associated descriptors (if they are specified), a list of events (presentation, selection or attribution) that need to be monitored by the media player, the presentation event that needs to be started (called here main-event), an optional offset-time and an optional delay-time.

The media object shall be derived from a <media> element, whose *src* attribute shall be used, by the media player, to locate the content and start its presentation. If the content cannot be located, or if the media player does not know how to handle the content type, the media player shall finish the starting operation without performing any action.

The descriptor shall be chosen by the formatter following the directives specified in the NCL document. If the *start* instruction results from a link action that has a descriptor explicitly declared in its <bind> element (*descriptor* attribute of the <link> element's children <bind> element), the resulting descriptor shall merge the attributes of the bind descriptor with the attributes of the descriptor specified in the corresponding <media> element. For the common attributes, the <bind> descriptor information shall superpose the <media> descriptor data. If the <bind> element does not contain an explicit descriptor, the descriptor evaluated by the formatter shall be the <media> descriptor, if this attribute was specified. Otherwise, a default descriptor for that *type* of <media> shall be chosen by the formatter.

The list of events to be monitored by a media player should also be computed by the formatter, taking into account the NCL document specification. It shall check all links where the media object and the resulting descriptor participate. When computing the events to be monitored, the formatter shall take into account the media-object perspective, i.e., the path of <body> and <context> elements to reach the <media> element. Only links contained in these <body> and <context> elements should be considered to compute the monitored events.

The offset-time parameter is optional and it has "zero" as its default value, and it is meaningful only for continuous media or static media with explicit duration. In this case, this parameter defines a time offset from the beginning (beginning-time) of the main-event, from which the presentation of the main-event shall be immediately started (that is, it commands the player to jump to the beginning-time + offset-time). Obviously, the offset-time value shall be lower than the main-event duration.

If the offset-time is greater than zero, the media player shall put the main-event in the *occurring* state, but the event *starts* transition shall not be notified. If the offset-time is zero, the media player shall put the main-event in the *occurring* state and notify the *starts* transition occurrence. Events that would have their end-times previous to the beginning-time of the main-event and events that would have their beginning times after the end-time of the main-event do not need to be monitored by the media player (the formatter should do this verification when building the monitored event list).

Monitored events that would have beginning-times before the beginning-time of the main-event and end-times after the start time (beginning-time + offset-time) of the main-event shall be put in the *occurring* state, but their *starts* transitions shall not be notified (links that depend on this transition shall not be fired). Monitored events that would have their end times after the main-event beginning-time, but before the start time (beginning-time + offset-time) shall have their *occurrences* attribute incremented but the *starts* and *stops* transitions shall not be notified. Monitored events that have beginning-times before the start time (beginning-time + offset-time) and end time after the start time shall be put in the *occurring* state, but the corresponding *starts* transition shall not be notified.

© ABNT 2011 - All rights reserved

The delay-time is also an optional parameter and its default value is “zero” too. If greater than zero, this parameter contains a time to be waited by the media player before starting the presentation.

If a media player receives a *start* instruction for an object already being presented (paused or not), it shall ignore the instruction and keep on controlling the ongoing presentation. In this case, the <simpleAction> element that has caused the *start* instruction shall not cause any transition on the corresponding event state machine.

8.2.2 Stop instruction for presentation events

The *stop* instruction only needs to identify a media object already being controlled. To identify the media object means to identify the <media> element and the corresponding descriptors. Therefore, if a <simpleAction> element with an *actionType* attribute equal to “stop” is bound through a link to a node interface, the interface shall be ignored when the action is performed.

If the object is not being presented (none of the events in the object list of events is in the *occurring* or *paused* state) and the media player is not waiting due to a delayed *start* instruction, the *stop* instruction shall be ignored. If the object is being presented, the main-event (the event passed as a parameter when the media object was started) and all monitored events in the *occurring* or in the *paused* state with end time equal or previous to the main-event end time shall transit to the *sleeping* state, and its *stop* transition shall be notified.

Monitored events in the *occurring* or in the *paused* state with end time posterior to the main-event end time shall be put in the *sleeping* state, but their *stops* transitions shall not be notified and their *occurrences* attribute shall not be incremented. The object content presentation shall be stopped. If the *repetitions* event attribute is greater than zero, it shall be decremented by one and the main-event presentation shall restart after the repeat delay time (the repeat delay shall have been passed to the media player as the start delay parameter). If the media object is waiting to be presented after a delayed *start* instruction and a *stop* instruction is issued, the previous *start* instruction shall be removed.

NOTE When all media objects refering to the elementary stream that carries the service main video are in the sleeping state the main video is dimensioned to 100 % of the screen. The main video can be redimensioned only using a media object in presentation. The same happens with the main audio. When all media objects refering to the elementary stream that carries the service main audio are in the sleeping state the main audio is presented with 100 % of its volume.

8.2.3 Abort instruction for presentation events

The *abort* instruction only needs to identify a media object already being controlled. If a <simpleAction> element with an *actionType* attribute equal to “abort” is bound through a link to a node interface, the interface shall be ignored when the action is applied.

If the object is not being presented and is not waiting to be presented after a delayed *start* instruction, the *abort* instruction shall be ignored. If the object is being presented, the main-event and all monitored events in the *occurring* or in the *paused* state shall transit to the *sleeping* state, and their *aborts* transitions shall be notified. Any content presentation shall stop.

If the *repetitions* event attribute is greater than zero, it shall be set to zero and the media object presentation shall not restart. If the media object is waiting to be presented after a delayed *start* instruction and an *abort* instruction is issued, the previous *start* instruction shall be removed.

8.2.4 Pause instruction for presentation events

The *pause* instruction only needs to identify a media object already being controlled. If a <simpleAction> element with an *actionType* attribute equal to “pause” is bound through a link to a node interface, the interface shall be ignored when the action is applied.

If the object is not being presented (the main-event, passed as a parameter when the media object was started, is not in the *occurring* state) and the media player is not waiting for the start delay, the instruction shall be ignored. If the object is being presented, the main-event and all monitored events in the *occurring* state shall transit to the *paused* state and their *pauses* transitions shall be notified. The object presentation shall be paused and the pause

© ABNT 2011 - All rights reserved

elapsed time shall not be considered as part of the object duration. As an example, if an object has an explicit duration of 30 s and, after 25 s it is paused, even if the object stays paused for 5 min, after resuming the object main-event shall stay occurring for 5 s. If the main-event is still not occurring because the media player is waiting for the start delay, the media object shall wait for a resume instruction to continue waiting for the remaining start delay.

8.2.5 Resume instruction for presentation events

The *resume* instruction only needs to identify a media object already being controlled. If a <simpleAction> element with an *actionType* attribute equal to “resume” is bound through a link to a node interface, the interface shall be ignored when the action is applied.

If the object is not paused (the main-event, passed as a parameter when the media object was started, is not in the *paused* state) or the media player is not paused (waiting for the start delay), the instruction shall be ignored.

If the media player is paused waiting for the start delay, it shall resume the wait from the instant it was paused. If the main-event is in the *paused* state, the main-event and all monitored events in the *paused* state shall be put in the *occurring* state and their *resumes* transitions shall be notified.

8.2.6 Start instruction for attribution events

The *start* instruction may be applied to an object property independent from the fact if the object is being presented or not (in this last case, although the object is not being presented, its media player shall be already instantiated). In both cases, the *start* instruction needs to identify the media object being controlled, a monitored attribution event, a value to be assigned to the attribute wrapped by the event, the duration of the attribution and the attribution step.

When setting a value to the attribute, the media player shall set the event state machine to the *occurring* state, and after finishing the attribution, again to the *sleeping* state, generating the *starts* transition and afterwards the *stops* transition.

For every monitored attribution event, if the media player changes by itself the corresponding attribute value, it shall also proceed as if it had received an external *start* instruction.

8.2.7 AddEvent instruction

The *addEvent* instruction is issued in the case of receiving an *addInterface* NCL editing command (see Section 9). The instruction needs to identify a media object already being controlled and a new event that shall be included to be monitored.

All rules applied to the intersection of monitored events with the main-event shall be applied to the new event. If the new event start time is previous to the object current time and the new event end time is posterior to the object current time, the new event shall be put in the same state of the main-event (*occurring* or *paused*), without notifying the corresponding transition.

8.2.8 RemoveEvent instruction

The *removeEvent* instruction is also issued in the case of receiving a *removeInterface* NCL editing command. The instruction needs to identify a media object already being controlled and a monitored event that should be no more controlled. The event state shall be put in the *sleeping* state without generating any transition.

8.2.9 Natural end of a presentation

Events of an object, with an explicit or an intrinsic duration, normally end their presentations naturally, without needing external instructions. In this case, the media player shall transit the event to the *sleeping* state and notify the *stops* transition. The same shall be done for monitored events in the *occurring* state with the same end time of the main-event or with unknown end time, when the main-event ends. Events in the *occurring* state with end time

© ABNT 2011 - All rights reserved

posterior to the main-event end time shall be put in the sleeping state but without generating the *stops* transition and without incrementing the *occurrences* attribute. It is important to remark that if the main-event corresponds to an object internal temporal anchor, when this anchor presentation finishes, the whole media object presentation shall finish.

8.3 Expected behavior of media players after instructions applied to composite objects

NOTE The concepts provided in this subclause also applies to a <media> element of "application/x-ginga-NCL" type that will behave as if it is a composite node made up by its <body> element and shall be treated accordingly.

8.3.1 Binding a composite node

A <simpleCondition> or <simpleAction> with *eventType* attribute value equal to "presentation" may be bound by a link to a composite node (represented by a <context> or <body> element) as a whole (i.e. without an interface being informed). As usual, the event state machine of the presentation event defined on the composite node shall be controlled as specified in 7.2.8.

Analogously, an <attributeAssessment> with *eventType* attribute value equal to "presentation" and *attributeType* equal to "state", "occurrences" or "repetitions" may be bound by a link to a composite node (represented by a <context> or <body> element) as a whole, and the attribute value should come from the event state machine of the presentation event defined on the composite node.

If a <simpleAction> with *eventType* attribute value equal to "presentation" is bound by a link to a composite node (represented by a <context> or <body> element) as a whole (i.e. without an interface being informed), the instruction shall also be reflected to the event state machines of the composite child nodes, as explained in the following subsections.

8.3.2 Starting a context presentation

If a <context> or <body> element participates on an action role whose action type is "start", when this action is fired, the *start* instruction shall also be applied to all presentation events mapped by the <context> or <body> element's ports.

If the author wants to start the presentation using a specific port, it shall in addition indicate the <port> id as the <bind> *interface* value.

8.3.3 Stopping a context presentation

If a <context> or <body> element participates on an action role whose action type is "stop", when this action is fired, the *stop* instruction shall also be applied to all presentation events of the composite child nodes.

If the composite node contains links being evaluated (or with their evaluation paused), the evaluations shall be suspended and no action shall be fired.

8.3.4 Aborting a context presentation

If a <context> or <body> element participates on an action role whose action type is "abort", when this action is fired, the *abort* instruction shall also be applied to all presentation events of the composite child nodes.

If the composite contains links being evaluated (or with their evaluation paused), the evaluations shall be suspended and no action shall be fired.

© ABNT 2011 - All rights reserved

8.3.5 Pausing a context presentation

If a <context> or <body> element participates on an action role whose action type is “pause”, when this action is fired, the *pause* instruction shall also be applied to all presentation events of the composite child nodes that are in the occurring state.

If the composite contains links being evaluated, all evaluations shall be suspended until a resume, stop or abort action is issued.

If the composite contains child nodes with presentation events already in the paused state when the pause action is issued, these nodes shall be identified because if the composite receives a resume instruction, these events shall not be resumed.

8.3.6 Resuming a context presentation

If a <context> or <body> element participates on an action role whose action type is “resume”, when this action is fired, the *resume* instruction shall also be applied to all presentation events of the composite child nodes that are in the paused state, except those that were already paused before the composite has been paused.

If the composite contains links with paused evaluations, they shall be resumed.

8.4 Relation between the presentation-event state machine of a node and the presentation-event state machine of its parent-composite node

NOTE The concepts provided in this subclause also applies to a <media> element of “application/x-ginga-NCL” type that will behave as if it is a composite node made up by its <body> element and shall be treated accordingly.

Whenever the presentation event of a node (media or composite) goes to the occurring state, the presentation event of the composite node that contains the node shall also enter in the occurring state.

When all child nodes of a composite node have their presentation events in the sleeping state, the presentation event of the composite node shall also be in the sleeping state.

Composite nodes do not need to infer *aborts* transitions from their child nodes. These transitions in presentation events of composite nodes shall occur only when instructions are applied directly to the composite node presentation event (see 8.3).

When all child nodes of a composite node have their presentation events in a state different from the *occurring* state and at least one child node have its main-event in the *paused* state, the presentation event of the composite node shall also be in the paused state.

If a <switch> element is started, but it does not define a default component and none of the <bindRule> referred rules is evaluated as true, the switch presentation shall not enter in the occurring state.

8.5 Expected behavior of imperative media players in NCL applications

Imperative objects may be inserted into NCL documents mainly to bring additional computational capabilities to declarative documents. The way to add a imperative object into an NCL document is to define a <media> element, whose content (located through the *src* attribute) is the imperative code to be executed. Both EDTV and BDTV profiles of NCL 3.0 allow two media types to be associated with the <media> element: application/x-ginga-NCLua, for Lua imperative codes (file extension .lua); and application/x-ginga-NCLet, for Java (Xlet) imperative codes (file extension .class or .jar).

Authors may define NCL links to start, stop, pause, resume or abort the execution of a imperative code. A imperative player (the language engine) shall interface the imperative execution environment with the NCL formatter.

© ABNT 2011 - All rights reserved

Analogous to conventional media content players, imperative players shall control event state machines associated with the NCL imperative node (NCLua or NCLet). As an example, if the code finishes its execution, the player shall generate the stops transition in the event presentation state machine corresponding to the imperative execution.

NCL allows imperative code execution to be synchronized with other NCL objects (imperative or not). A <media> element containing a imperative code may also define anchors (through <area> elements) and properties (through <property> elements).

A imperative code span may be associated with an <area> element (using the *label* attribute). If external links start, stop, pause, resume or abort the anchor presentation, callbacks in the imperative code span shall be triggered. The way these callbacks are defined is responsibility of each imperative code associated with the NCL imperative object.

On the other hand, a imperative code span may also command the start, stop, pause or resume of its <area> elements through an API offered by the imperative language. These transitions may be used as conditions of NCL links to trigger actions on other NCL objects of the same document. Thus, a two-way synchronization can be established between the imperative code and the remainder of the NCL document.

The other way a imperative code may be synchronized with other NCL objects is through <property> elements. A <property> element defined as a child of a <media> element representing a imperative code may be mapped to a code span (function, method, etc.) or to a code attribute. When it is mapped to a code span, a link action “set” applied to the property shall cause the code span execution, with the set values interpreted as input parameters. The *name* attribute of the <property> element shall be used to identify the imperative code span. When the <property> element is mapped to a imperative code attribute the action “set” shall assign the value to the attribute. As usual, the event state machine associated with the property shall be controlled by the imperative player.

A <property> element defined as a child of a <media> element representing a imperative code may also be associated with an NCL link assessment role. In this case, the NCL formatter shall query the property value in order to evaluate the link expression. If the <property> element is mapped to a code attribute, the code attribute value shall be returned by the imperative player to the NCL formatter. If the <property> element is mapped to a code span, it shall be called and the output value resulting from its execution shall be returned by the imperative player to the NCL formatter.

The *start* instruction issued by a formatter shall inform the following parameters to the imperative player: the locator of the imperative object to be controlled, its associated descriptors, a list of events (defined, by the <media> element’s <area> and <property> child elements) that need to be monitored by the imperative player, the <area> element *id* associated with the imperative code to be started, and an optional delay-time. From the *src* attribute, the imperative player tries to locate the imperative code and start its execution. If the content cannot be located, the imperative player shall finish the starting operation, without performing any action.

The list of events to be monitored by a imperative player should also be computed by the formatter, taking into account the NCL document specification. It shall check all links where the imperative object and the resulting descriptor participate. When computing the events to be monitored, the formatter shall take into account the imperative media-object perspective, that is, the path of <body> and <context> elements to reach the <media> element. Only links contained in these <body> and <context> elements should be considered to compute the monitored events.

As with any other <media> element, the delay-time is an optional parameter and its default value is “zero”. If greater than zero, this parameter contains a time to be waited by the imperative player before starting the code execution.

Different from what is performed on other <media> elements, if a imperative player receives a *start* instruction for an event associated with an <area> element and this event is in the *sleeping* state, it shall start the execution of the imperative code span associated with the element, even though other portion of the object’s imperative code is being in execution (paused or not). However, if the event associated with the target <area> element is in the *occurring* or *paused* state, the *start* instruction shall be ignored by the imperative code player that keeps on controlling the ongoing execution. As a consequence, different from what happens for other <media> elements, a <simpleAction> element with an *actionType* attribute equal to “stop”, “pause”, “resume” or ”abort” shall be bound through a link to a imperative node interface, which shall not be ignored when the action is applied.

© ABNT 2011 - All rights reserved

The *start* instruction issued by a formatter for an event associated with a <property> element may be applied to an imperative object independent from the fact whether it is being executed or not (in the latter case, although the object is not being executed, its imperative player shall have already been instantiated). In both cases, the *start* instruction needs to identify the imperative object in execution, a monitored attribution event, and a value to be passed to the imperative code wrapped by the event.

For every monitored attribution event, if the imperative player changes by itself the corresponding attribute value, it shall also proceed as if it had received an external *start* instruction.

Imperative languages should also offer an API that allows imperative code to query any pre-defined or dynamic properties' values of the NCL settings node (<media> element of "application/x-ginga-settings" type). However, it must be stressed that it is not allowed to directly set values to these properties. Properties of the application/x-ginga-settings node may only be changed trough using NCL links.

9 Content transmission and NCL events

9.1 Private bases

The core of the Ginga-NCL presentation engine is composed of the NCL Formatter and its Private Base Manager module.

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*. Ginga associates a private base with a TV channel. 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).

This section deals only with editing commands coming from the terrestrial broadcast channel.

NOTE The same editing commands may also come from the interactive channel or from events generated by NCLua or NCLet imperative objects.

Ginga adopts specific MPEG-2 sections (identified by the tableID field of the MPEG-2 Section) to transport editing commands in MPEG-2 TS elementary streams, when commands come from the terrestrial broadcast channel.

Editing commands are wrapped in a structure called *event descriptors*. Event descriptors have a structure composed basically of an id (identification), a time reference and a private data field. The identification uniquely identifies each editing command event.

The time reference indicates the exact moment to trigger the event. A time reference equal to zero informs that the event shall be triggered immediately after being received (events carrying this type of time-reference are commonly known as "do it now" events). The private data field provides support for event parameters (see Figure 5).

© ABNT 2011 - All rights reserved

Syntax	Number of bits
EventDescriptor ( ) {	
eventId	16
eventNPT	33
privateDataLength	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 to 1928
FCS	8
}	

Figure 5 – Editing command event descriptor

The *commandTag* uniquely identifies the editing commands, as specified in Table 56. In order to allow sending a complete command in more than one event descriptor, all 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*.

DSM-CC stream events can be used to transport event descriptors. 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. Event objects are used to inform Ginga about DSM-CC stream events that can be received. Event names allow specifying types of events, offering a higher abstraction level for middleware applications. The Private Base Manager, as well as NCL execution-objects (for example. NCLua, NCLet), should, in this case, register themselves as listeners of stream events they handle, using event names.Besides event objects, the DSM-CC object carousel protocol is also used to transport files organized in directories. The Ginga DSM-CC demultiplexer is responsible for mounting the file system at the receiver device (see Clause 12 and ABNT NBR 15606-3).

NCL document files and NCL media-object's contents are organized in file system structures. XML-based editing *command parameters* may be directly transported in the payload of an event descriptor or, alternatively, organized in file system structures to be transported, each one, in the datacasting channel, or still be received from the interactivity channel.

EXAMPLE DSM-CC stream events can be used to transport event descriptors. The DSM-CC object carousel protocol allows the cyclical transmission of stream event objects and file systems. Stream event objects are used to map stream event names into stream event ids. Stream event objects are used to inform Ginga about DSM-CC stream events that can be received. Event names allow specifying types of events, offering a higher abstraction level for middleware applications. The Private Base Manager, as well as NCL execution-objects (for example, NCLua, NCLet), should, in this case, register themselves as listeners of stream events they handle, using event names.

The received *commandTag* is used by the Private Base Manager to interpret the complete *command string* semantics. 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 relates a set of file paths with their respective location in the transport system. In the case of pulled files received from an interactivity channel or 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.

© ABNT 2011 - All rights reserved

Table 56 shows the *command strings* and, surrounded by round brackets, the parameters carried as the payload content of an *nclEditingCommand* event descriptor. The commands are divided into three groups: the first one for private base operation (to open, close, activate, deactivate, and save private bases); the second one for document manipulation (to add, remove, and save a document in an open private base and to start, pause, resume, and stop document presentations in an active private base); and the last one for NCL entities handling in an open private base. For each NCL entity, *add* and *remove* commands are defined. If an entity already exists, the *add* command has the update (altering) semantics.

NOTE The first group of commands for private base operation do not usually come from the terrestrial broadcast channel. As aforementioned, editing commands may also come from the interactive channel or from events generated by NCLua or NCLet imperative objects. Editing commands can also be internally generated by the middleware.

Table 56 – Editing commands for Ginga's private base manager

Command string	Command tag	Description
openBase (baseId, location)	0x00	Opens an existing private base located with the *location* parameter. If the private base does not exist or the *location* parameter is not informed, a new base is created with the *baseId* identifier. The location parameter shall specify the storage device in the receiver environment and the path for opening the base
activateBase (baseId)	0x01	Turns on an open private base. All applications are then enabled to be started.
deactivateBase (baseId)	0x02	Turns off an open private base. All application shall be stopped.
saveBase (baseId, location)	0x03	Saves all private base content into a persistent storage device (if available). The *location* parameter shall specify the device and the path for saving the base
closeBase (baseId)	0x04	Closes the opened private base and disposes all private base content
addDocument (baseId, {uri, id}+)	0x05	Adds an NCL document to an open private base. The NCL document's files can be: a) 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 Section 12); 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. b) received from an interactivity channel 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 baseId, if this NCL document is not received as a pushed file
removeDocument (baseId, documentId)	0x06	Removes an NCL document from an open private base

© ABNT 2011 - All rights reserved

Table 56 (continuação)

Command string	Command tag	Description
startDocument (baseId, documentId, interfaceId, offset,nptBaseId, nptTrigger) NOTE The offset parameter is a time value. NOTE The nptTrigger is an NPT value and the nptBaseId is an NPT time base identifier.	0x07	Starts playing an NCL document in an active private base, beginning the presentation from a specific document interface. The time reference provided in the nptTrigger field defines the initial time positioning of the document with regards to the NPT time base identified in the *nptBaseId field* Three cases may happen: a) If nptTrigger is different from 0 and is greater than or equal to the current NPT value of the NPT time base identified by nptBaseId, the document presentation shall wait until NPT has the value specified in nptTrigger to be started from its beginning time+*offset*. b) If nptTrigger is different from 0 and is less than the current NPT value of the NPT time base identified by nptBaseId, the document shall be started immediately from its beginning time+*offset*+$(NPT - nptTrigger)_{seconds}$ NOTE Only in this case, the *offset* parameter value may be a negative time value, but *offset*+$(NPT - nptTrigger)_{seconds}$ shall be a positive time value. c) If nptTrigger is equal to 0, the document shall start its presentation imediatelly from its beginning time+*offset*
stopDocument (baseId, documentId)	0x08	Stops the presentation of an NCL document in an active private base. All document events that are occurring shall be stopped
pauseDocument (baseId, documentId)	0x09	Pauses the presentation of an NCL document in an active private base. All document events that are occurring shall be paused
resumeDocument (baseId, documentId)	0x0A	Resumes the presentation of an NCL document in an active private base. All previously document events that were paused by the pauseDocument editing command shall be resumed.
saveDocument (baseId, documented, location)	0x2E	Saves an NCL document of an opened private base into a persistent storage device (if available). The location parameter shall specify the device and the path for saving the document. If the NCL document to be saved is running in the opened 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 a transport system as a file system, the *xmlRegionBase* parameter is just a reference to this content
removeRegionBase (baseId, documentId, regionBaseId)	0x0E	Removes a <regionBase> element from the <head> element of an NCL document in a private base

© ABNT 2011 - All rights reserved

Table 56 (continuação)

Command string	Command tag	Description
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 a transport system as a file system, the *xmlRuleBase* parameter is just a reference to this content
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 a transport system as a file system, the *xmlConnectorBase* parameter is just a reference to this content
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 *xmlDescriptorSwitch* 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 *xmlDescriptorBase* parameter is just a reference to this content
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

© ABNT 2011 - All rights reserved

Table 56 – Editing commands for Ginga's private base manager

Command string	Command tag	Description
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 *xmlTransitionBase* 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) of an NCL document in a private base
removeImportBase (baseId, documentId, docBaseId, documentURI)	0x22	Removes an <importBase> element, whose *documentURI* attribute is identified by the documentURI parameter, from the base (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase>, or <connectorBase> element) 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, documentId, documentURI)	0x26	Removes an <importNCL> element, whose *documentURI* 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	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: a) sent in the datacast network as a set of pushed files; the {uri,id} pair is used to relate file paths in the XML document specification of the node with their respective locations in a transport system (see examples in Section 12); 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 locations in the receiver memory. b) received from an interactivity channel as s 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

© ABNT 2011 - All rights reserved

Table 56 – Editing commands for Ginga's private base manager

Command string	Command tag	Description
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 *name* attribute or a <port>, <area>, or <switchPort> element's *id* 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
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 *propertyId* shall identify a <property> element's *name* attribute or a <switchPort> element's *id* 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

Receivers that only implement the NCL Basic DTV profile cannot handle the following commands: *pauseDocument*, *resumeDocument*, *addTransition*, *removeTransition*, *addTransitionBase* and *removeTransitionBase*.

Ginga associates at least one private base with each TV channel. When a channel is tunned, its corresponding base is opened and activated by the Private Base Manager; other private bases shall be deactivated. For security reasons, only one private base may be active at a time. The simplest and most restricted way to manage private bases is having only one private base opened at a time. Thus, if the user changes the selected channel, the current private base should be closed. In this case, the *openBase* command is always followed by the *activateBase* command, and the *deactivateBase* command is never used. However, the number of private bases that may be kept opened is a specific middleware implementation decision.

Add commands always have NCL entities as their arguments (XML-based command parameters). Whether the specified entity already exists or not, document consistency shall be maintained by the NCL formatter, in the sense that all entity attributes classified as required shall be defined. The entities are defined using a syntax notation identical to that used by the NCL schemas, with the exception of the *addInterface* command: the *begin* attribute of an <area> element may receive the "now" value, specifying the current NPT of the nodeId, which shall be the main MPEG video being played by the hardware decoder.

The identifiers used in the commands shall be in agreement with Table 57.

© ABNT 2011 - All rights reserved

Table 57 – Identifiers used in editing commands

Identifiers	Definition
baseId	Broadcast channel identifiers specified by the SBTVD
documentId	The *id* attribute of an <ncl> element of an NCL document
nptTrigger	A value of NPT
nptBaseId	The *contentId* identifier of an NPT time base
regionId	The *id* attribute of a <region> element of an NCL document
ruleId	The *id* attribute of a <rule> element of an NCL document
connectorId	The *id* attribute of a <connector> element of an NCL document
descriptorId	The *id* attribute of a <descriptor> element of an NCL document
descriptorSwitchId	The *id* attribute of a <descriptorSwitch> element of an NCL document
transitionId	The *id* attribute of a <transition> element of an NCL document
regionBaseId	The *id* attribute of a <regionBase> element of an NCL document
ruleBaseId	The *id* attribute of a <ruleBase> element of an NCL document
connectorBaseId	The *id* attribute of a <connectorBase> element of an NCL document.
descriptorBaseId	The *id* attribute of a <descriptorBase> element of an NCL document
transitionBaseId	The *id* attribute of a <transitionBase> element of an NCL document
docBaseId	The *id* attribute of a <regionBase>, <ruleBase>, <connectorBase>, <descriptorBase>, or <transitionBase> element of an NCL document
documentURI	The *documentURI* attribute of an <importBase> element or an <importNCL> element of an NCL document
importedDocumentBaseId	The *id* attribute of a <importedDocumentBase> element of an NCL document
compositeID	The *id* attribute of a <body>, <context> or <switch> element of an NCL document
nodeId	The *id* attribute of a <body>, <context>, <switch> or <media> element of an NCL document
interfaceId	The *id* attribute of a <port>, <area>, <property> or <switchPort> element of an NCL document
linkId	The *id* attribute of a <link> element of an NCL document
propertyId	The *id* attribute of a <property> or <switchPort> element of an NCL document

9.2 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. Receivers that only implement the NCL Basic DTV profile should ignore the XML elements and attributes related to Meta-information and Transitions functionalities.

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

© ABNT 2011 - All rights reserved

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/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"/>
 <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"
```

© ABNT 2011 - All rights reserved

```
    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"/>
```


© ABNT 2011 - All rights reserved

```
        <element ref="profile:property"/>
        <element ref="profile:switchPort"/>
        <element ref="profile:link"/>
        <element ref="profile:meta"/>
        <element ref="profile:metadata"/>
      </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>
```

© ABNT 2011 - All rights reserved

```
      <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"/>
```


© ABNT 2011 - All rights reserved

```
      <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">
```

© ABNT 2011 - All rights reserved

```
    </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>
```


© ABNT 2011 - All rights reserved

```
  <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>
```

© ABNT 2011 - All rights reserved

```
    <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>
  </complexContent>
</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>
```

 © ABNT 2011 - All rights reserved

```
  <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>
```

© ABNT 2011 - All rights reserved

```
  <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>
```


© ABNT 2011 - All rights reserved

10 Lua imperative objects in NCL presentations

10.1 Lua language - Removed functions in the Lua library

The scripting language adopted by Ginga-NCL to implement imperative objects in NCL documents is *Lua* (<media> elements of application/x-ginga-NCLua type or application/x-ncl-NCLua type). The complete definition of Lua is presented in Annex B.

The following functions are platform dependent and were removed in the implementation:

a) in module *package*: *loadlib*;

b) in module os: clock, execute, exit, getenv, remove, rename, tmpname and setlocale;

c) in module *debug*: all functions.

10.2 Execution model

The lifecycle of an NCLua object is controlled by the NCL formatter. The formatter is responsible for triggering the execution of an NCLua object and for mediating the communication among an NCLua object and other nodes in an NCL document, as defined in Clause 8.5.

As with all media object players, once instantiated, the Lua player shall execute an initialization procedure. However, different from other media players, this initialization code is specified by the NCLua object author. This initialization procedure is executed only once, for each instance, and creates functions and objects that may be used during the NCLua object execution and, in particular, registers one (or more) event handler for communication with the NCL formatter.

After the initialization, the execution of the NCLua object becomes event oriented in both directions. That is, any action commanded by the NCL formatter reaches the registered event handlers, and any NCL event state change notification is sent as an event to the NCL formatter (as for example, the natural end of a procedure execution). The Lua Player is then ready to perform any *start* or *set* instruction (see 8.5).

10.3 Additional modules

10.3.1 Required modules

Besides the Lua standard library, the following modules shall be implemented and automatically loaded:

a) module canvas: offers an API to draw graphical primitives and manipulate images;

b) module event: allows NCLua applications to communicate with the middleware through events (NCL, pointer and key events);

c) module settings: exports a table with variables defined by the NCL document author and reserved environment variables contained in an "application/x-ginga-settings" node;

d) module persistent: exports a table with persistent variables, which may be manipulated only by imperative objects.

The definition of each function in the above modules use the following naming convention:

funcname (parname1: partype1 [; optname1: opttype1]) -> retname: rettype

© ABNT 2011 - All rights reserved

10.3.2 *Canvas* module

10.3.2.1 *Canvas* object

When an NCLua media object is initialized, the corresponding region of the <media> element (of type application/x-ginga-NCLua) is available as the global *canvas* variable for the Lua script. If the <media> element has no associated region defined (*left, right, top and bottom* properties), then the value for *canvas* is set to “nil”.

As an example, assume an NCL document region defined as:

```
<region id="luaRegion" width="300" height="100" top="200" left="20"/>
```

The `canvas` variable in a NCLua media object referring to “luaRegion” is bound to a canvas object of size 300x100, associated with the specified region at (20,200).

A canvas offers a graphical API to be used in an NCLua application. Using the API, it is possible to draw lines, rectangles, font, images, etc.

A canvas keep in its state a set of attributes under which the drawing primitives operate. For instance, if its color attribute is blue, a call to `canvas:drawLine()` will draw a blue line on the canvas.

The coordinates are always relative to the top/leftmost point in canvas (0,0).

10.3.2.2 Constructors

From any canvas object, it is possible to create new canvas and combine them through composite operations.

---

canvas:new (image_path: string) -> canvas: object

Arguments

image_path	Image path

Return values

canvas	Canvas representing the image

Description

Retorns a new canvas whose content is the image received as a parameter.

The new canvas shall keep the transparency aspects of the original image.

---

canvas:new (width, height: number) -> canvas: object

Arguments

width	Canvas width
height	Canvas height

Return values

canvas	New canvas

© ABNT 2011 - All rights reserved

Description

Returns a new canvas with the received size.

Initially, all pixels shall be transparent.

10.3.2.3 Attributes

All attribute methods have the prefix “attr” and are used to get and set attributes (with the exceptions specified).

When a method is invoked without input parameters, the current attribute value is returned. On the other hand, when a method is invoked with input parameters, these parameters must be used as the new attribute values.

---

canvas:attrSize () -> width, height: number

Arguments

Return values

width	Canvas width
height	Canvas height

Description

Returns the canvas dimensions.

It is important to note that it is not possible to change the dimensions of an existing canvas.

---

canvas:attrColor (R, G, B, A: number)

Arguments

R	Color red component
G	Color green component
B	Color blue component
A	Color alpha component

Description

Change canvas’ attribute color.

The colors are given in RGBA, where A varies from 0 (full transparency) to 255 (full opacity).

The primitives (see 10.3.3.4) are drawn with the color set to this attribute.

The initial value is ‘0,0,0,255’ (black).

© ABNT 2011 - All rights reserved

canvas:attrColor (clr_name: string)

Arguments

clr_name	Color name

Change canvas' attribute color.

The colors are given as a string corresponding to one of the 16 pre-defined NCL colors:

'white', 'aqua', 'lime', 'yellow', 'red', 'fuchsia', 'purple', 'maroon',

'blue', 'navy', 'teal', 'green', 'olive', 'silver', 'gray', 'black'

The values given have their alpha equal to full opacity ("A = 255").

The primitives (see 10.3.3.4) are drawn with the color set in this attribute.

The initial value is 'black'.

canvas:attrColor () -> R, G, B, A: number

Return values

R	Color red component
G	Color green component
B	Color blue component
A	Color alpha component

Description

Retorns the canvas' color.

canvas:attrFont (face: string; size: number; style: string)

Arguments

face	Font name
size	Font size
style	Font style

Description

Changes canvas' font attribute.

The following fonts shall be available: 'Tiresias'.

The size is in pixels, and it represents the maximum height of a line written with the chosen font.

The possible style values are: 'bold', 'italic', 'bold-italic' and 'nil'. A 'nil' value assumes that no style will be used.

Any invalid input value shall raise an error.

The initial font value is undefined.

© ABNT 2011 - All rights reserved

canvas:attrFont () -> face: string; size: number; style: string

Return values

face	Font name, or the path and the name of the font type
size	Font size
style	Font style

Description

Returns the canvas font.

canvas:attrClip (x, y, width, height: number)

Arguments

x	Clipping area coordinate
y	Clipping area coordinate
width	Clipping area width
height	Clipping area height

Description

Changes the canvas clipping area.

The drawing primitives (see 10.3.3.4) and the method `canvas:compose()` only operate inside this clipping region.

The initial value is the whole canvas.

canvas:attrClip () -> x, y, width, height: number

Return values

x	Clipping area coordinate
y	Clipping area coordinate
width	Clipping area width
height	Clipping area height

Description

Returns the canvas clipping area.

© ABNT 2011 - All rights reserved

canvas:attrCrop (x, y, w, h: number)

Arguments

x	Crop region coordinate
y	Crop region coordinate
w	Crop region width
h	Crop region height

Description

Changes the canvas *crop* region.

Only the set region is affected by operations following graphical compositions.

The initial *crop* region is the whole canvas.

The main canvas cannot have its *crop* region changed as it is controlled by the NCL formatter.

canvas:attrCrop () -> x, y, w, h: number

Return values

x	Crop region coordinate
y	Crop region coordinate
w	Crop region width
h	Crop region height

Description

Returns the canvas *crop* region.

canvas:attrFlip (horiz, vert: boolean)

Arguments

horiz	If canvas should be flipped horizontally
vert	If canvas should be flipped vertically

Description

Sets the canvas flipping mode used when the canvas is composed.

The main canvas cannot be flipped as it is controlled by the NCL formatter.

© ABNT 2011 - All rights reserved

canvas:attrFlip () -> horiz, vert: boolean
Return values

horiz	If canvas is flipped horizontally
vert	If canvas is flipped vertically

Description

Returns the current canvas' flipping setup.

canvas:attrOpacity (opacity: number)

Argument

opacity	Canvas opacity

Description

Changes canvas opacity.

The opacity values varies between 0 (full transparency) to 255 (full opacity).

The main canvas cannot have its value changed as it is controlled by the NCL formatter.

canvas:attrOpacity () -> opacity: number

Return value

opacity	Canvas opacity

Description

Returns the current canvas opacity.

canvas:attrRotation (degrees: number)

Argument

degrees	Canvas rotation in degrees.

Description

Sets the canvas rotation attribute that must be multiple of 90^{o}.

The main canvas cannot have its value changed as it is controlled by the NCL formatter.

canvas:attrRotation () -> degrees: number
Return value

degrees	Canvas rotation in degrees

Description

Returns the current canvas rotation value.

© ABNT 2011 - All rights reserved

canvas:attrScale () -> w, h: number|boolean

Arguments

w	Canvas scaling width
h	Canvas scaling height

Description

Scales the canvas to a given width and height.

One of the given values may be *true,* indicating that the aspect ratio must be kept.

The scaling attribute is independent of the size attribute, which shall remain the same.

The main canvas cannot have its value changed as it is controlled by the NCL formatter.

canvas:attrScale () -> w, h: number

Return values

w	Canvas scaling width
h	Canvas scaling height

Description

Returns the current canvas scaling values.

10.3.2.4 Primitives

All the following methods take the canvas' attributes into account.

canvas:drawLine (x1, y1, x2, y2: number)

Arguments

x1	Line extremity 1 coordinate
y1	Line extremity 1 coordinate
x2	Line extremity 2 coordinate
y2	Line extremity 2 coordinate

Description

Draws a line with its extremities in (x1,y1) and (x2,y2).

© ABNT 2011 - All rights reserved

canvas:drawRect (mode: string; x, y, width, height: number)

Arguments

mode	Drawing mode
x	Rectangle coordinate
y	Rectangle coordinate
width	Rectangle width
height	Rectangle height

Description

Method for rectangle drawing and filling.

The parameter mode may receive 'frame' or ‘fill’ values, for drawing the rectangle with no-fill or filling it, respectively.

canvas:drawRoundRect (mode: string; x, y, width, height, arcWidth, arcHeight: number)

Arguments

mode	Drawing mode
x	Rectangle coordinate
y	Rectangle coordinate
width	Rectangle width
height	Rectangle height
arcWidth	Rounded edge arc width
arcHeight	Rounded edge arc height

Description

Function for rounded rectangle drawing and filling.

The parameter *mode* may be *'frame'* in order to draw the rectangle frame or *'fill'* to fill it.

canvas:drawPolygon (mode: string) -> drawer: function

Arguments

mode	Drawing mode

Return values

f	Drawing function

Description

Method for polygon drawing and filling.

© ABNT 2011 - All rights reserved

The parameter mode may receive the 'open' value, to draw the polygon not linking the last point to the first; the 'close' value, to to draw the polygon linking the last point to the first; or the 'fill' value, to draw the polygon linking the last point to the first and painting the region inside.

The function canvas:drawPolygon returns an anonymous function “drawer” with the signature:

function (x, y) end

The returned function, receives the next polygon vertex coordinates and returns itself as the result. This recurrent procedure allows the idiom:

canvas:drawPolygon('fill')(1,1)(10,1)(10,10)(1,10)()

When the function "drawer" receives ‘nil’ as input, it completes the chained operation. Any subsequent call shall raise an error.

---

canvas:drawEllipse (mode: string; xc, yc, width, height, ang_start, ang_end: number)

Arguments

mode	Drawing mode
xc	Ellipse center
yc	Ellipse center
width	Ellipse width
height	Ellipse height
ang_start	Starting angle
ang_end	Ending angle

Description

Draws an ellipse and other similar primitives as circle, arcs and sectors.

The parameter mode may receive ‘arc’ to only draw the circunference or ‘fill’ for internal painting.

---

canvas:drawText (x,y: number; text: string)
Arguments

x	Text coordinate
y	Text coordinate
text	Text do be drawn

Description

Draws the given text at (x,y) in the canvas, using the font set by `canvas:attrFont()`.

© ABNT 2011 - All rights reserved

10.3.2.5 Miscellaneous

canvas:clear ([x, y, w, h: number])

Arguments

x	Clear area coordinate
y	Clear area coordinate
w	Clear area width
h	Clear area height

Description

Clears the canvas with the color set to *attrColor*.

If the area parameters are not given, all the canvas should be cleared.

canvas:flush ()

Description

Flushes the canvas after a set of drawing and composite operations.

It's enough to call this method only once, after a sequence of operations.

canvas:compose (x, y: number; src: canvas; [src_x, src_y, src_width, src_height: number])

Arguments

x	Position of the composition
y	Position of the composition
src	Canvas to compose with
src_x	Position in the canvas src
src_y	Position in the canvas src
src_width	Composition width in the canvas src
src_height	Composition height in the canvas src

Description

Composes pixel by pixel the canvas src on the current canvas (destination canvas) at position (x,y).

The other parameters are optionals and indicate which region in the canvas src is used to compose with. When absent the whole canvas is used.

This operation calls src:flush() automatically before the composition.

© ABNT 2011 - All rights reserved

The operation satisfies the following equation:

Cd = Cs*As + Cd*(255 - As)/255

Ad = As*As + Ad*(255 - As)/255

where:

Cd = color of the destination canvas (canvas)

Ad = alpha of the destination canvas (canvas)

Cs = color of the source canvas (src)

As = alpha of the source canvas (src)

After the operations the destination canvas has the resulting content and the canvas src remains intact.

---

canvas:pixel (x, y, R, G, B, A: number)

Arguments

x	Pixel position
y	Pixel position
R	Color red component
G	Color green component
B	Color blue component
A	Color alpha component

Description

Changes the pixel color.

---

canvas:pixel (x, y: number) -> R, G, B, A: number

Arguments

x	Pixel position
y	Pixel position

Return values

R	Color red component
G	Color green component
B	Color blue component
A	Color alpha component

Description

Returns the pixel color.

© ABNT 2011 - All rights reserved

canvas:measureText (text: string) -> dx, dy: number

Arguments

text	Text to be measured

Return values

dx	Text width
dy	Text height

Description

Returns the border coordinates for the given text, as if it were drawn at (x,y) with the configured font of `canvas:attrFont()`.

10.3.3 The *event* module

10.3.3.1 General view

This module offers an API for event handling. Using the API, the NCL formatter may communicate with an NCLua application asynchronously.

An application may also use this mechanism internally, using the “user” event class.

The typical use of NCLua application is to handle events: NCL events (see Section 7.2.8) or events coming from user interactions (for example, through the remote control).

During its initiation, before becoming event oriented, a Lua script has to register an event handler function. After the initialization any action performed by the script will be in response to an event notified to the application, that is, to the event handler function.

```
=== example.lua ===

...             -- initializing code

function handler (evt)

  ...           -- handler code

end

event.register(handler)   -- register as an event listener

=== end ===
```

Among the event types that may be received by the handler function are all those generated by the NCL formatter. As aforementioned, a Lua script is also capable of generating events, called “spontaneous”, trough a call to the `event.post(evt)` function.

© ABNT 2011 - All rights reserved

10.3.3.2 Functions

event.post ([dst: string]; evt: event) -> sent: Boolean; err_msg: string

Arguments

dst	Event destination
evt	Event to be posted

Return values

sent	If the event was successfully sent
err_msg	Error message in case of errors

Description

Posts the given event.

The parameter "dst" is the event destination and may assume the values "in" (send to itself) and "out" (send to the NCL formatter). The default value is ‘out’.

event.timer (time: number, f: function) -> cancel: function

Arguments

time	Time in milliseconds
f	Callback function

Return value

unreg	Function to cancel the timer

Description

Creates a timer that expires after a timeout (in milliseconds) and then call the callback function f.

The signature of f is simple, no parameters are received or returned:

```
function f () end
```

The value of 0 milliseconds is valid. In this case, event.timer() shall return immediately and `f` shall be called as soon as possible.

© ABNT 2011 - All rights reserved

event.register ([pos: number]; f: function; [class: string]; […: any])
Arguments

pos Register position (optional)

f Callback function

class Class filter (optional)

… Class dependent filter (optional)

Description

Registers the given function as an event listener, that is, whenever an event happens, `f` is called (the function `f` is an event handler).

The parameter *pos* is optional.It indicates the position where *f* is registered. If it is not given, the function is registered in the last position.

The parameter *class* is optional and indicates which class of events the function shall receive. If *class* is specified, other class dependent filters may be defined. A *nil* value in any position indicates that the parameter shall not be filtered.

The signature for `f` is:"

```
function f (evt) end -> handled: boolean
```

Where `evt` is the event that triggers the function.

The function may return "true", to signalize that the event was handled and, therefore, should not be sent to other handlers.

event.unregister (f: function)
Arguments

f Callback function

Description

Unregisters the given function as a listener, that is, new events will no longer be notified to `f`.

event.uptime () -> ms: number
Return values

ms Time in milliseconds

Description

Returns the number of milliseconds elapsed since the beginning of the application.

© ABNT 2011 - All rights reserved

10.3.3.3 Event classes

The function `event.post()` and the registered handler in `event.register()` receive events as parameters.

An event is described by a common Lua table, where the class field is mandatory and identifies the event class.

The following event classes are defined:

`key` class:

evt = { class='key', type: string, key: string}

* type may be 'press' or 'release'.

* key is the key value; the "event.keys" table holds all keycodes available in the NCL.

Example evt = { class='key', type='press', key='0'}

NOTE In the key class, the class dependent filter could be *type* and *key*, in this order.

`pointer` Class:

evt = { class='pointer', type: string, x=number, y=number }

* type may be 'press', 'release' or 'move'

* x and y refer to the coordinates of the pointer event occurrence

NOTE In the pointer class, the class dependent filter could only be *type*.

EXAMPLE evt = { class='pointer', type='press', x=20, y=50 }

`ncl` class:

Relations among NCL media nodes are based on events. Lua has access to these events through `ncl class`.

Events may act in two directions, that is, the formatter may send action events to change the state of the Lua player, which in its turn may trigger transition events to signal state changes.

In events, the type field shall assume one of the three values:
'presentation', 'selection' or 'attribution'

Events may be directed to specific anchors or to the whole node, this is identified by the label field, that assumes the whole node when absent.

In the case of an event generated by the formatter, the action field shall have one of the following values: 'start', 'stop', 'abort', 'pause' or 'resume'

Type 'presentation':

evt = { class='ncl', type='presentation', label='?', action='?' }

Type 'attribution':

© ABNT 2011 - All rights reserved

evt = { class='ncl', type='attribution', name='?', action='?', value='?' }

For events generated by the Lua player, the "action" field shall assume one of the following values:
'start', 'stop', 'abort', 'pause ,or 'resume'

Type 'presentation':

evt = { class='ncl', type='presentation', label='?', action='start'/'stop'/'abort'/'pause'/'resume' }

Type 'selection':

evt = { class='ncl', type='selection', label='?', action='stop' }

Type 'attribution':

evt = { class='ncl', type='attribution', name='?', action='start'/'stop'/'abort'/'pause'/'resume', value='?' }

NOTE In the ncl class, the class dependent filter could be *type*, *label*, and *action*, in this order.

---

edit class:

This class reproduces the editing commands for the Private Base manager (see Clause 9). However, there is an important difference between editing commands coming from DSM-CC stream events (see Clause 9), and the editing commands performed by Lua scripts (NCLua objects). The first ones alter not only the NCL document presentation, but also the NCL document specification. That is, in the end of the process a new NCL document is generated incorporating all editing results. On the other hand, editing commands coming from NCLua media objects only alter the NCL document presentation. The original document is preserved during all editing process.

Just like in other event classes, an editting command is represented by a Lua table. All events shall contain the *command* field: a string with the command name. The other fields depend on the command type(see Table 56 in Section 9). The unique difference is with regards to the field that defines the {uri,id} reference pairs, named *data* field in the edit class. This field's values may be not only the reference pairs mentioned in Table 56, but also XML strings with the content to be added.

EXEMPLE

```
evt = {
    command = 'addNode',
    compositeId = 'someId',
    data = '<media>...',
}
```

The *baseId* e *documentId* fields are optional (when applicable) and they assume by default the base and document identifiers where the NCLua object is in execution.

The event describing the editting command may also receive a time reference as an optional parameter (optional parameters are indicated in the function signatures as arguments between brackets). This optional parameter may be used to specify the exact moment when the editing command shall be executed. If this parameter is not

© ABNT 2011 - All rights reserved

provided in the function call, the editing command shall be executed immediately. When provided, this parameter may have two different types of values, with two different meanings. If it is a number value, it defines the amount of time, in seconds, for how long the command shall be postponed. However, this parameter may also specify the exact moment, in absolute values, the command shall be executed. In this case, this parameter shall be a table value with the following fields: year (four digits), month (1-12), day (1-31), hour (0-23), min (0-59), sec (0-61), and isdst (a daylight saving flag, a boolean).

---

tcp class:

The use of the return channel is done through this class of events.

In order to send or receive a tcp data, a connection shall be firstly established trough posting an event in the form:

evt = { class='tcp', type='connect', host=addr, port=number, [timeout=number] }

The connection result is returned in a pre-registered event handler for the class. The returned event is in the form:

evt = { class='tcp', type='connect', host=addr, port=number, connection=identifier, error=<err_msg>}

The *error* and *connection* fields are mutually exclusive. When there is a communication error, a message is returned in the error field. When the communication is succeeded, the connection identifier is returned in the *connection* field.

An NCLua application sends data, using a return channel, through posting events in the form:

evt = { class=’tcp’, type='data', connection=identifier, value=string, [timeout=number] }

Similarly, an NCLua application receives data transported in a return channel using events in the form:

evt = { class=’tcp’, type=’data’, connection=identifier, value=string, error=msg}

The *error* and *value* fields are mutually exclusive. When there is a communication error, a message is returned in the *error* field. When the communication is succeeded, the message is passed in the *value* field.

In order to close the connection, the following event shall be posted:

evt = { class='tcp', type='disconnect', connection=identifier }

NOTE 1 A specific middleware implementation should handle issues like authentication.

NOTE 2 In the tcp class, the class dependent filter could only be *connection*.

---

sms class:

An NCLua application sends data, using SMS, through posting events in the form:

evt = { class=’sms’, type=’send’, to=’string’, value=string [, id:string]}

The `to` field contains the destination number (phone number or large account number). If they are not specified, region and country code prefixes will receive the respective region and country codes from where the message is being sent.

The `value` field contains the message content.

The `id` field can be used to identify the SMS that will be dispatched. The application is responsible for defining the `id` value and guarantee its unicity.

© ABNT 2011 - All rights reserved

A confirmation event shall be sent back to the NCLua application, following the format:

evt = { class='sms', type='send', to:string, sent:Boolean [,error:string] [, id:string] }

In the confirmation message the `to` field shall have the same value as in the original event posted by the NCLua application. The `sent` field notify if the SMS was dispatched by the device (true) or not. The `error` field is optional. If the `sent` field value is false, it may contain a detailed error message. If the original SMS is posted with the `id` field defined, the confirmation event shall arrive with the same id value. Thus, the NCLua application will be able to make an association between both events, and deal with multiple SMS messages being dispatched simultaneously.

Similarly, an NCLua application registers itself to receive SMS messages by posting events in the form:

evt = { class='sms', type ='register', port:number }

The `port` field shall receive a valid TCP port number. For compliance with the GSM Standards (3GPP TS 23.040 V6.8.1, of 2006-10), this value should be in the interval [16000,16999].

Events received by the handler have the following format:

evt = { class='sms', type='receive', from:string, port:number, value:string }

The `port` field is defined as in the type = 'register'. The `from` field contains the source message number (phone number or large account number). Region and country code prefixes may be omitted if they are equal to the receiver ones. The `value` field contains the message content.

At any moment, the application can request to stop receiving SMS messages in a given port, posting the event:

evt = { class='sms', type='unregister', port:number }

The port field is defined as in the type = 'register'.

At the moment the NCLua media presentation stops, the middleware implementation shall ensure that all ports will be unregistered.

NOTE 1 An specific middleware implementation should handle issues like authentication, etc.

NOTE 2 In the sms class, the class dependent filter could only be *from* and *port*, in this order.

NOTE 3 The purpose of the port number is to avoid conflicts between common SMS messages received by a user, and SMS messages that are to be handle only by the application.

A Ginga-NCL implementation shall immediately return *false* in every call to event.post() that uses an event class that is not supported. The NCLua application should capture this error condition in order to verify if the SMS dispatch failed.

---

si class:

The `si` event class provides access to a set of information multiplexed in a transport stream and periodically transmitted.

The information acquisition process shall be performed in two steps:

1) A request is made calling the asynchronous `event.post()` function;

2) An event, to be delivered to the registered-event handlers of an NCLua script, whose data field contains a set of subfields and is represented by a Lua table. The set of subfields depends on requested information.

NOTE In the si class, the class dependent filter could only be *type*.

Four event types are defined by the following tables:

type = 'services'

The table of 'services' event type is made up by a set of vectors, each one with information related with a multiplexed service of the tuned transport stream.

© ABNT 2011 - All rights reserved

Each request for a table of ‘services’ event type shall be carried out through the following call:

event.post('out', { class='si', type='services'[, index=N][, fields={field_1, field_2,…, field_j}]}),

where:

a) the *index* field defines the service index, when specified; if not specified, all services of the tuned transport stream shall be present in the returned event;

b) the *fields* table may have as a value any subset of subfields defined for the *data* table of the returned event (thus, field_i represents one of the subfields of the data table, as defined in what follows). If the *fields* list is not specified, all subfields of the data table shall be filled.

The returned event is created after all request information is processed by the middleware (information that is not broadcasted within the maximum interval specified by ABNT NBR 15603-2:2007, Table 6, shall be returned as ‘nil’). The data table is returned as follows:

```
evt = {
   class = 'si',
   type = 'services',
   data = {
      [i] = {  -- each service for each i
         id                          = <number>,
         isAvailable                 = <boolean>,
         isPartialReception          = <boolean>,
         parentalControlRating       = <number>,
         runningStatus               = <number>,
         serviceType                 = <number>,
         providerName                = <string>,
         serviceName                 = <string>,
         stream = {
            [j] = {
               pid = <number>,
               componentTag = <number>,
               type = <number>,
               regionSpecType = <number>,
               regionSpec     = <string>,
            }
      }
   }
}
```

In order to compute the values of the data-table subfields to be returned in events of services type, SI tables should be used as a basis, as well as descriptors associated with the service [i].

© ABNT 2011 - All rights reserved

The values of the *id* and *runningStatus* data-table subfields should be computed according to the values of service_id and running_status fields, respectively, of the SDT table (see ABNT NBR 15603-2:2007, Table 13) that describes the service [i].

The values of the *providerName* and *serviceName* data-table subfields should be computed according to the values of *service*_name and service_provider_name fields, respectively, of the service_descriptor (see ABNT NBR 15603-2:2007) that describes the service [i].

The value of the *parentalControlRating* data-table subfield should be computed according to the value of the rating field of the parental_rating_descriptor that has the country_code field with the equivalent country value that has the user.location variable of the Settings node.

The value of the *isAvaiable* data-table subfield should be computed according to the value of the country_code field (with the available set of countries) of the country_availability_descriptor (see ABNT NBR 15603-2:2007, 8.3.6) related with service [i]. The “true” value shall be assigned only if the country_code field has a country value equivalent to the value of the user.location variable of the Settings node.

The value of the *isPartualReception* data-table subfield should be computed according to the value of service_id field of the partial_reception_descriptor (see ABNT NBR 15603-2:2007, 8.3.32).

The semantics of the *serviceType* data-table subfield should be defined by ABNT NBR 15603-2:2007, Table H.2.

The semantics of the *runningStatus* data-table subfield should be defined by ABNT NBR 15603-2:2007, Table 14.

The value of the *pid* stream-table subfield should have the same value of the pid field of the elementary stream [i] header (see ISO/IEC 13818-1).

The value of the *componentTag* stream-table subfield should be computed according to the value of component_tag field of the stream_identifier_descriptor (see ABNT NBR 15603-2:2007, 8.3.16) related with the elementary stream [i].

The semantics of the *type* stream-table subfield should be defined according to ISO/IEC 13818-1: 2008, Table 2-34, related with the elementary stream [i].

The coding method for the *regionSpec* stream-table subfield should be defined by *regionSpecType* stream-table subfield, according to the semantics defined in ABNT NBR 15603-2:2007, Table 53.

The value of the *regionSpec* stream-table subfield should define the region for which the elementary stream [i] is designated.

The *regionSpec* and *regionSpecType* stream-table subfields should also be computed based on the target_region_descriptor (see ABNT NBR 15603-2:2007).

type = ‘mosaic’

The table of the ‘mosaic’ event type is made up by a set of information for building the mosaic, and is provided in a matrix format. This table is optional.

When the table of the mosaic event type is provided, each request for a table of ‘mosaic’ event type shall be carried out through the following call:

event.post('out', { class='si', type='mosaic'[, fields={field_1, field_2,…, field_j}]}),

where the *fields* list may have as a value any subset of subfields defined for the *data* table of the returned event (thus, field_i represents one of the subfields of the data table, as defined in what follows). If the *fields* list is not specified, all subfields of the data table shall be filled.

The returned event is created after all request information is processed by the middleware (information that is not broadcasted within the maximum interval specified by ABNT NBR 15603-2:2007, Table 6, of shall be returned as ‘nil’). The data table is returned as follows:

© ABNT 2011 - All rights reserved

```
evt = {
   class = 'si',
   type = 'mosaic',
   data = {
      [i] = {
         [j] = {
            logicalId       = <number>,
            presentationInfo = <number>,
            id              = <number>,
            linkageInfo     = <number>,
            bouquetId       = <number>,
            networkId       = <number>,
            tsId            = <number>,
            serviceId       = <number>,
            eventId         = <number>,
         }
      }
   }
}
```

NOTE In order to compute the values of the data-table subfields to be returned in events of mosaic type, SI tables should be used as a basis, as well as descriptors associated with the mosaic. The maximum values for [i] and [j], as well as the values of the logicalId, presentationInfo, id, linkageInfo, bouquetId, networkId, tsId, serviceId and eventId data-table subfields should be computed according to the values of number_of_horizontal_elementary_cells, number_of_vertical_elementary_cells, logical_cell_id, logical_cell_presentation_info, id, cell_linkage_info, bouquet_id, original_network_id, transport_stream_id, service_id and event_id fields, respectively of the mosaic_descriptor (See ABNT NBR 15603-2:2007, 8.3.9).

type = ‘epg’

The table of the ‘epg’ event type is made up by a set of vectors. Each vector contains information about an event of the content being transmitted.

Each request for a table of ‘epg’ event type shall be carried out through one of the following possible calls:

1) event.post('out', { class='si', type='epg', stage=’current’[, fields={field_1, field_2,…, field_j}]})

where the *fields* list may have as a value any subset of subfields defined for the *data* table of the returned event (thus, field_i represents one of the subfields of the data table, as defined in what follows). If the *fields* list is not specified, all subfields of the data table shall be filled.

Description: returns information regarding to the current event of the content being transmitted.

2) event.post('out', {class='si', type='epg', stage='next'[, eventId=<number>][, fields={field_1, field_2,…, field_j}]})

where:

© ABNT 2011 - All rights reserved

a) the eventid field, when specified, identifies the event immediately before the event whose information is required. When not specified, the requested information is for the event that immediately follows the curresnt event.

b) the *fields* list may have as a value any subset of subfields defined for the *data* table of the returned event (thus, field_i represents one of the subfields of the data table, as defined in what follows). If the *fields* list is not specified, all subfields of the data table shall be filled.

Description: returns information regarding to the event immediately after the event defined in *eventId*, or information regarding to the event immediately after the current event, when *eventId* is not specified.

3) event.post('out', {class='si', type='epg', stage='schedule', startTime=<date>, endTime=<date>[, fields={field_1, field_2,…, field_j}]})

where the *fields* list may have as a value any subset of subfields defined for the *data* table of the returned event (thus, field_i represents one of the subfields of the data table, as defined in what follows). If the *fields* list is not specified, all subfields of the data table shall be filled.

Description: returns information regarding to events within the time interval defined by the startTime and endTime fields, which have tables in the <date> format as values.

The returned event is created after all request information is processed by the middleware (information that is not broadcasted within the maximum interval specified by ABNT NBR 15603-2:2007, Table 6, of shall be returned as 'nil'). The data table is returned as follows:

```
evt = {
   class = 'si',
   type = 'epg',
   data = {
      [i] – {
         startTime              = <date>,
         endTime                = <date>,
         runningStatus          = <number>,
         name                   = <string>,
         originalNetworkId      = <number>,
         shortDescription       = <string>,
         extendedDescription    = <string>,
         copyrightId            = <number>,
         copyrightInfo          = <string>,
         parentalRating         = <number>,
         parentalRatingDescription = <string>,
         audioLanguageCode      = <string>,
         audioLanguageCode2     = <string>,
         dataContentLanguageCode = <string>,
         dataContentText        = <string>,
         hasInteractivity       = <boolean>,
         logoURI                = <string>,
```

© ABNT 2011 - All rights reserved

```
contentDescription       = {
   [1] = <content_nibble_1>,
   [2] = <content_nibble_2>,
   [3] = <user_nibble_1>,
   [4] = <user_nibble_2> }
},
linkage = {
   tsId      = <number>,
   networkId = <number>,
   serviceId = <number>,
   type      = <number>,
   data      = <string>,
},
hyperlink = {
   type           = <number>,
   destinationType = <number>,
   tsId           = <number>,
   networkId      = <number>,
   eventId        = <number>,
   componentTag   = <number>,
   moduleId       = <number>,
   serviceId      = <number>,
   contentId      = <number>,
   url            = <string>,
},
series = {
   id               = <number>,
   repeatLabel      = <number>,
   programPattern   = <number>,
   episodeNumber    = <number>,
   lastEpisodeNumber = <number>,
   name = <string>,
},
eventGroup = {
   type = <number>,
   [j] = {
      id       = <number>,
      tsId     = <number>,
      networkId = <number>,
```


© ABNT 2011 - All rights reserved

```
            serviceId = <number>,
         }
      },
      componentGroup = {
         type = <number>,
         [j] = {
            id          = <number>,
            totalBitRate = <number>,
            description  = <string>,
            caUnit = {
               id       = <number>,
               component = {
                  [k] = tag (<number>)
               }
            },
         }
      }
   }
}
```

In order to compute the values of the data-table subfields to be returned in events of epg type, SI tables should be used as a basis, as well as descriptors associated with the event [i].

The values of the *startTime, endTime, runningStatus* and *originalNetworkId* data-table subfields should be computed according to the values of the start_time, (duration + start_time), running_status and original_network_id fields, respectively, of the SI table event_information_section (see ABNT NBR 15603-2:2007, Table 15).

The values of the *name* and *shortDescription* data-table subfields should be computed according to the values of the event_name_char and text_char fields, respectively, of the short_event_descriptor (see ABNT NBR 15603-2:2007, 8.3.15).

The value of the *extendedDescription* data-table subfield should be computed according to the value of the text_char field of the extended_event_descriptor (see ABNT NBR 15603-2:2007, 8.3.7).

The values of the *copyrightId* e *copyrightInfo* data-table subfields should be computed according to the values of the copyright_identifier and additional_copyright_info fields, respectively, of the copyright_descriptor (see ISO/IEC 13818-1: 2008, Table 2-63).

The semantics of the *parentalRating* data-table subfield should be defined according to ABNT NBR 15603-2:2007, Table 33. Its value should be computed according to the value of the country_code field of the parental_rating_descriptor and the environment variable (Settings node) user.location.

The semantics of the *parentalRatingDescription* data-table subfield should be defined according to ABNT NBR 15603-2:2007, Table 32. Its value should be computed according to the value of the country_code field of the parental_rating_descriptor and the environment variable (Settings node) user.location.

© ABNT 2011 - All rights reserved

The values of the *audioLanguageCode* and *audioLanguageCode2* data-table subfields should be computed according to the values of the ISO_639_language_code and text_char fields, respectively, of the data_content_descriptor (see ABNT NBR 15603-2:2007, Table 54).

The values of the *dataContentLanguageCode* and *dataContextText* data-table subfields should be computed according to the values of the ISO_639_language_code and text_char fields, respectively, of the data_content_descriptor (see ABNT NBR 15603-2:2007, Table 54).

The value of the *hasInteractivity* data-table subfield shall have the “true” value when event [i] has an interactive application available.

The value of the *logoURI* data-table subfield should define the logotype location transmitted in a CDT Table (see ABNT NBR 15603-2:2007, 8.3.44).

The subfield values of the *contentDescription* table should be computed according to corresponding fields of the content_descriptor (See ABNT NBR 15603-2:2007, 8.3.5).

The values of the *tsId, networkId, serviceId, type* and *data* linkage-table subfields should be computed according to the values of the transport_stream_id, original_network_id, original_service_id, description_type and user_defined fields, respectively, of the linkage_descriptor (see ABNT NBR 15603-2:2007, 8.3.40).

The values of the *type, destinationType, tsId, networkId, eventId, componentTag, moduleId, contentId* and *url* hyperlink-table subfields should be computed according to the values of the hyper_linkage_type, link_destination_type, transport_stream_id, original_network_id, event_id, component_tag, moduleId, content_id and url_char fields, respectively, of the hyperlink_descriptor (see ABNT NBR 15603-2:2007, 8.3.29).

The values of the *id, repeatLabel, programPattern, episodeNumber, lastEpisodeNumber* and *name* series-table subfields should be computed according to the values of the series_id, repeat_label, program_pattern, episode_number, last_episode_number and series_name_char fields, respectively, of the series_descriptor (see ABNT NBR 15603-2:2007, 8.3.33).

The values of the *type, id, tsId, networkId* and *serviceId* eventGroup-table subfields should be computed according to the values of the group_type, event_id, transport_stream_id, original_network_id and service_id fields, respectively, of the event_group_descriptor (see ABNT NBR 15603-2:2007, 8.3.34).

The values of the *type, id, totalBitRate, description, caUnit.id, caUnit.component[k].tag, tsId, networkId* and *serviceId* componentGroup-table subfields should be computed according to the values of the component_group_type, component_group_id, total_bit_rate, text_char, CA_unit_id and component_tag fields, respectively, of the component_group_descriptor (see ABNT NBR 15603-2:2007, 8.3.37).

type=’time’

The table of the ‘time’ event type contains information about the current UTC (Universal Time Coordinated) date and time, but in the official country time zone in which the receptor is located.

Each request for a table of ‘time’ event type shall be carried out through the following call:

```
event.post('out', { class='si', type=’time’})
```

The returned event is created after all request information is processed by the middleware (information that is not broadcasted within the maximum interval specified by ABNT NBR 15603-2:2007, Table 6, shall be returned as ‘nil’). The data table is returned as follows:

```
evt = {
   class = 'si',
   type = 'time',
```


© ABNT 2011 - All rights reserved

```
data = {
        year        = <number>,
        month       = <number>,
        day         = <number>,
        hours       = <number>,
        minutes     = <number>,
        seconds     = <number>
}
```

NOTE In order to compute the values of the data-table subfields to be returned in events of time type, the TOT table should be used as a basis, as well as the local_time_offset_descriptor, according to ABNT NBR 15603-2:2007, 7.2.9.

---

user class:

By using the class user, applications may extend their functionalities, create their own events.

In this class, no fields are defined (with the exception of the class field).

NOTE In the user class, the class dependent filter could be *type*, if this field is defined.

10.3.4 *Settings* module

Exports the *settings* table with the reserved environment variables and the variables defined by the NCL document author, as defined in the application/x-ginga-settings node.

It is not allowed to set values to the fields representing variables in the settings node. An error shall be raised in this case. Properties of the application/x-ginga-settings node may only be changed trough using NCL links.

The settings table splits its groups into several subtables, corresponding to each application/x-ginga-settings node's group. For instance, in an NCLua object, the settings node's variable "system.CPU" is referred to as settings.system.CPU.

Examples of use:

```
lang = settings.system.language

age  = settings.user.age

val  = settings.default.selBorderColor

settings.service.myVar = 10

settings.user.age = 18 --> ERROR!
```

When referring an indexed property of the application/x-ginga-settings object, the index (i) shall be replaced by [i] in the NCLua object.

© ABNT 2011 - All rights reserved

10.3.5 Persistent module

NCLua applications may save data in a restricted middleware area and recover it between executions. Lua player allows an NCLua application to persist a value to be used by itself or by another imperative object. In order to do that it defines a reserved area, inaccessible to non-imperative NCL media objects. This area is split into the groups “service”, “channel” and “shared”, with same semantics of the homonym groups of the NCL settings node. There are no predefined or reserved variables in these groups, and imperative objects are allowed to change variable’s values directly. Other imperative languages, in particular Java for NCLet objects (<media> elements of type application/x-ginga-NCLet) should offer an API to access this same area.

In this module, Lua offers an API to export the *persistent* table with the variables defined in the reserved area.

The use of the *persistent* table is very similar to the *settings* table, except that, in this case, imperative codes may change field values.

Examples of use:

persistent.service.total = 10

color = persistent.shared.color

10.4 Lua-API for Ginga-J

10.4.1 Mapping

Depending on the middleware configuration, it is possible to have access in Lua to the same API provided by Ginga-J, in order to have access to some set-top box resources and Ginga facilities. The API provided in Lua is optional, but when provided it shall follow the same specification defined for Ginga-J, and thus Clause 10.4 only describes how the Java API provided by Ginga-J is mapped to Lua.

10.4.2 Packages

The hierarchies of Java packages that compose the Ginga-J API are mapped to equivalent hierarchies of Lua packages that have a common root package, called ginga. More specifically, a package “x” in Ginga-J API is mapped to an equivalent Lua package ginga.x. In this context, an *equivalent* Lua package means a package that contains classes and sub-packages equivalent to those defined in the Java package.

The set of Ginga-J packages that will be available in the execution environment of a Lua script may be restricted by security policies. If a package “x” of the Ginga-J API is available in the Lua environment, ginga.x will hold a reference to a Lua table with all definitions related to “x” (classes and sub-packages). Otherwise, ginga.x is a nil reference. Some examples of name mappings of Ginga-J packages to Lua packages are presented in Table 58.

© ABNT 2011 - All rights reserved

Table 58 – Examples of name mappings between Ginga-J packages and Lua packages

Ginga-J package	Lua package
org.sbtvd.net.tuning	ginga.org.sbtvd.net.tuning
org.sbtvd.media	ginga.org.sbtvd.media
javax.media	ginga.javax.media
org.dvb	ginga.org.dvb
org.havi	ginga.org.havi
org.davic	ginga.org.davic

10.4.3 Basic types

The Java's basic data types, used in the Ginga-J API, are mapped to Lua's basic data types. These mappings shall be in agreement with Table 59. In addition to the primitive types of Java, this table also specifies the mapping of strings and arrays.

Table 59 – Mapping of basic data types

Java type	Lua type
short	number
int	number
long	number
float	number
double	number
byte	number
char	string (with only one character)
boolean	boolean
Array objects	table
String objects	string

10.4.4 Classes

Every Java class of the Ginga-J API is represented in Lua as a table, defined in its respective package. For instance, the class

org.sbtvd.net.tuning.ChannelManager

is represented in Lua as an entry ChannelManager in the package ginga.org.sbtvd.net.tuning, that is, this class is accessed through

ginga.org.sbtvd.net.tuning.ChannelManager.

All static members of a Java class are mapped to fields of the equivalent Lua table. Every class represented in Lua also has a newInstance operation that plays the role of a *constructor*.

10.4.5 Objects

Every time the newInstance method provided by a class represented in Lua is called, it returns a new instance (*object*) of that class. The returned object is a Lua table that has all instance members specified by its class (public fields and methods).

10.4.6 Callback objects (listeners)

Many methods defined in the Ginga-J API expect to receive a *listener* object as parameter. These listener objects may be implemented in Lua as tables that have all methods specified in the listener's interface.

© ABNT 2011 - All rights reserved

10.4.7 Exceptions

Java exceptions are also mapped to Lua tables, following the same rules to map Java objects to Lua. To raise an exception, a listener object implemented in Lua should use the function error provided by Lua (see Annex B). To catch an exception raised by an API method, the Lua script should use the function pcall (see Annex B).

11 Bridge

11.1 Review

The two-way bridge between Ginga-NCL and Ginga-J is done:

— in one way, through NCL relationships, defined in <link> elements that refer to <media> elements representing Xlet (application/x-ginga-NCLet type) codes supported by Ginga-J; and through Lua scripts (<media> elements of the application/x-ginga-NCLua type) referencing Ginga-J methods;

— in the reverse way, through Ginga-J functions that may monitor any NCL event and may also command changes in NCL elements and properties, through relationships defined in <link> elements or through NCL editing commands.

11.2 Bridge through <link> and <media> NCL elements

As aforementioned, Ginga-NCL may act on Ginga-J through <link> elements and through <media> elements of the application/x-ginga-NCLet type.

Analogous to conventional media content, NCL allows Xlet code to be synchronized with other NCL objects (imperative or not). NCL authors may define NCL links to start, stop, pause, resume or abort the execution of an Xlet imperative code (represented by a <media> element of the application/x-ginga-NCLet type), as they do for usual presentation contents (see 8.5). An NCLet player (based on the Java engine) shall interface the imperative execution environment with the NCL formatter (see 8.5).

A <media> element containing a Java code may define anchors (through <area> elements) and attributes (through <property> elements). The player shall control the state machine of events associated with these interface elements.

Xlet code may be associated with an <area> element. If external links start, stop, pause or resume the anchor presentation, callbacks in the Xlet code shall be triggered. On the other hand, the Xlet code may command the start, stop, pause, resume or abort of these anchors through an API offered by the imperative language. The transitions caused by these commands may be used as conditions of NCL links to trigger actions on other NCL objects of the same document. Thus, a two-way synchronization may be established between the Xlet code and the remainder of the NCL document.

A <property> element defined as a child of a <media> element of the application/x-ginga-NCLet type may be mapped to an XLet-code method or to an Xlet-code attribute. When it is mapped to a code method, a link action “set” applied to the property shall cause the method execution, with the set values interpreted as parameters passed to the method. The *name* attribute of the <property> element shall be used to identify the imperative code method. When the <property> element is mapped to an Xlet code attribute, the action “set” shall assign a value to the attribute.

The <property> element may also be associated with an NCL link assessment role. In this case, the NCL formatter shall query the attribute value in order to evaluate the link expression. If the <property> element is mapped to a code attribute, the code attribute value shall be returned by the Xlet player to the NCL formatter. If the <property> element is mapped to a code method, the method shall be called and its output value shall be returned by the Xlet player to the NCL formatter.

© ABNT 2011 - All rights reserved

11.3 Bridge through Lua functions and Ginga-J methods

Depending on the middleware configuration, it is possible to have access in Lua to the same API provided by the Ginga-J, in order to have access to some set-top box resources and Ginga facilities. The API provided in Lua shall follow the same specification presented for Ginga-J.

Ginga-J also offers an API that allows Xlet code to query any pre-defined or dynamic properties' values of the NCL settings node (<media> element of "application/x-ginga-settings" type).

Moreover, Ginga-J offers NCL APIs that provides a set of methods to support NCL's editing commands and Private Base Manager commands.

12 Media coding requirements and transmission methods referred in NCL documents

12.1 Interactive channel use

An NCL formatter shall successfully ignore any coding or transmission method that is not supported by the browser. In order to acquire data content that is referred by <media> elements through a specific interactive channel protocol, the mechanisms specified for the interactive channel shall be used.

12.2 Video coding and transmission methods - Video data referred by <media> elements

12.2.1 Transmission of MPEG-1 video

12.2.1.1 Transmission as video elementary stream

To transmit MPEG-1 video content as a video elementary stream, the video data shall be transmitted as an MPEG-2 packetized elementary stream (video PES), with the stream type specified as the stream type assignment of ISO/IEC 13818-1 (value of 0x01 for ISO/IEC 11172-2).

12.2.1.2 Transmission in MPEG-2 sections

To transmit MPEG-1 video data through specific MPEG-2 sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1), one of the following transmission methods shall be used:

a) as a file of multiplexed stream in MPEG-1 systems (see ISO/IEC 11172-1);

b) as a file of MPEG-1 video elementary stream;

c) as a file of multiplexed stream in the TS format specified in 12.4.

12.2.2 Transmission of MPEG-2 video

12.2.2.1 Transmission as video elementary stream

To transmit MPEG-2 video content as a video elementary stream, the video data shall be transmitted as an MPEG-2 packetized elementary stream (video PES), with the stream type specified as the stream type assignment of ISO/IEC 13818-1 (value of 0x02 for ISO/IEC 13818-2 video).

12.2.2.2 Transmission in MPEG-2 sections

To transmit MPEG-2 video data through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1), one of the following transmission methods shall be used:

a) as a file of MPEG-2 video elementary stream;

b) as a file of multiplexed stream into the TS format specified in 12.4.

© ABNT 2011 - All rights reserved

12.2.3 Transmission of MPEG-4 video and H.264|MPEG-4 AVC

12.2.3.1 Transmission as video elementary stream

To transmit MPEG-4 video content as a video elementary stream, the video data shall be transmitted as an MPEG-2 packetized elementary stream (video PES), with the stream type specified as the stream type assignment of ISO/IEC 13818-1 (value of 0x10 for ISO/IEC 14496 video and H.264|MPEG-4 AVC).

12.2.3.2 Transmission MPEG-2 sections

To transmit MPEG-4 video or H.264|MPEG-4 AVC data through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1), one of the following transmission methods shall be used:

a) as a file of MPEG-4 video (or H.264|MPEG-4 AVC) elementary stream;

b) as a file of multiplexed stream into the TS format specified in 12.4.

12.3 Audio coding and transmission methods - Audio data referred by <media> elements

12.3.1 Transmission of MPEG-1 audio

12.3.1.1 Transmission as audio elementary stream

To transmit MPEG-1 audio content as an audio elementary stream, the audio data shall be transmitted as an MPEG-2 packetized elementary stream (audio PES), with the stream type specified as the stream type assignment of ISO/IEC 13818-1 (value of 0x03 for ISO/IEC 11172-3).

12.3.1.2 Transmission in MPEG-2 sections

To transmit MPEG-1 audio data through specific MPEG-2 sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1), one of the following transmission methods shall be used:

a) as a file of multiplexed stream in MPEG-1 systems (see ISO/IEC 11172-1);

b) as a file of MPEG-1 audio elementary stream;

c) as a file of multiplexed stream into the TS format specified in 12.4.

12.3.2 Transmission of MPEG-2 audio

12.3.2.1 Transmission as audio elementary stream

To transmit MPEG-2 AAC audio content as an audio elementary stream, the audio data shall be transmitted as an MPEG-2 packetized elementary stream (audio PES), with the stream type specified as the stream type assignment of ISO/IEC 13818-1 (value of 0x0F for ISO/IEC 13818-7).

To transmit MPEG-2 BC audio content as an audio elementary stream, the audio data shall be transmitted as an MPEG-2 packetized elementary stream (PES), with the stream type specified as the stream type assignment of ISO/IEC 13818-1 (value of 0x04 for ISO/IEC 13818-3).

12.3.2.2 Transmission in MPEG-2 sections

To transmit MPEG-2 audio data through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1), one of the following transmission methods shall be used:

a) As a file of MPEG-2 audio elementary stream;

b) As a file of multiplexed stream into the TS format specified in 12.4.

© ABNT 2011 - All rights reserved

12.3.3 Transmission of MPEG-4 audio

12.3.3.1 Transmission as audio elementary stream

To transmit MPEG-4 audio content as an audio elementary stream, the audio data shall be transmitted as an MPEG-2 packetized elementary stream (audio PES), with the stream type specified as the stream type assignment of ISO/IEC 13818-1 (value of 0x11 for ISO/IEC 14496-3).

12.3.3.2 Transmission in MPEG-2 sections

To transmit MPEG-4 audio data through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1), one of the following transmission methods shall be used:

a) As a file of MPEG-4 audio elementary stream;

b) As a file of multiplexed stream into the TS format specified in 12.4.

12.3.4 Transmission of AC3 audio

12.3.4.1 Transmission as audio elementary stream

To transmit AC3 audio content as an audio elementary stream, the audio data shall be transmitted as an MPEG-2 packetized elementary stream (audio PES), with the stream type specified as 0x81.

12.3.4.2 Transmission in MPEG-2 sections

To transmit AC3 audio data through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1), one of the following transmission methods shall be used:

a) As a file of AC3 audio elementary stream;

b) As a file of multiplexed stream into the TS format specified in 12.4.

12.3.5 Transmission of PCM (AIFF-C) audio

AIFF-C PCM audio should be transmitted as a file through specific MPEG-2 sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1).

12.4 TS format for MPEG video/audio transmission - Data encoding specification

12.4.1 Transmission of video and audio multiplexed

To transmit MPEG-1/2/4 Video or H.264|MPEG-4 AVC data along with MPEG-1/2/4 or AC3 Audio data in multiplexed files in specific MPEG-2 Sections, each multiplexed video/audio file is coded in a TS format as defined in ISO/IEC 13818-1.

12.4.2 Required PSI

A PAT shall be described. Any PAT shall be described with program_number whose value is other than 0 and that shall represent a PID of the PMT. The available values of program_number are defined in an operational standard regulation.

A PMT shall be described. Any stream identification descriptor indicating a second loop shall contain a PMT descriptor. Otherwise, a descriptor may be placed as required.

© ABNT 2011 - All rights reserved

It is recommended that the available values to component_tag, and the occurrence rules of component_tag in a default ES and PMT descriptors in a second loop are equivalent to an operational standard regulation used for the main stream of the media type responsible for transmitting the concerned stream.

In an implementation in which a transport stream is decoded from a file, which has been transmitted based on the data encoding specification defined in this section, is presented in a high-speed digital interface, an SIT shall be described (see ABNT NBR 15606-1). In other cases, SITs are not required, unless otherwise specified explicitly.

Any table other than PAT, PMT, and SIT (e.g. CAT, NIT, SDT, BAT, EIT, RST, TDT, TOT, PCAT, SDTT, and ST (see ABNT NBR 15601) shall not be described.

A PAT shall occur in a stream at a frequency of not less than one time per 100 milliseconds. A PMT shall occur in a stream at a frequency of not less than one time per 100 milliseconds.

As far as the single TS format file, a PAT/PMT shall not be modified or updated.

12.4.3 Transmission in MPEG-2 sections

To transmit a file coded with the data encoding specification defined in Section 12.6 in specific MPEG-2 sections, the transmission shall comply with the ABNT NBR 15606-3.

12.4.4 Constraints in playing

To perform receiving a broadcasting service and playing a TS file, received from specific MPEG-2 sections, two separate transport stream processing systems are required. The constraints in integrating and coordinating a content/event received via a broadcasting service and a TS file is not described in this Recommendation.

12.5 Coding scheme and transmission of still pictures and bitmap graphics data referred by <media> elements

12.5.1 Transmission of MPEG-2 I-frame, MPEG-4 I-VOP, and H.264|MPEG-4 AVC I-picture

12.5.1.1 Transmission in video PES for linear playback

To transmit a still picture in MPEG-2 I-frames through a video PES component, the coding scheme shall conform to the conventions defined in ABNT NBR 15606-1. The PES component shall be transmitted as a stream with the stream type value of 0x02.

To transmit a still picture in MPEG-4 I-VOP through a video PES component, the coding scheme shall conform to the conventions defined in ABNT NBR 15606-1. The PES component shall be transmitted as a stream with the stream type value of 0x10.

To transmit a still picture in H.264|MPEG-4 AVC I-picture through a video PES component, the coding scheme shall conform to the conventions defined in ABNT NBR 15606-1. The PES component shall be transmitted as a stream with the stream type value of 0x1B.

12.5.1.2 Transmission in MPEG-2 sections for interactive playback

To transmit a still picture in MPEG-2 I frames in MPEG-2 sections, the coding scheme shall conform to the conventions in ABNT NBR 15606-1. The still picture shall be transmitted as a file in the MPEG-2 section.

To transmit a still picture in MPEG4-I-VOP in MPEG-2 sections, the coding scheme shall conform to the conventions in ABNT NBR 15606-1. The still picture shall be transmitted as a file in the MPEG-2 section.

To transmit a still picture in H.264|MPEG-4 AVC I-picture in MPEG-2 sections, the coding scheme shall conform to the conventions in ABNT NBR 15606-1. The still picture shall be transmitted as a file in the MPEG-2 section.

In these cases, the stream type value of the MPEG-2 section shall be in agreement with ISO/IEC 13818-1.

© ABNT 2011 - All rights reserved

12.5.2 Transmission of JPEG still picture

JPEG still pictures shall be transmitted through specific MPEG-2 sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1).

12.5.3 Coding scheme and transmission of PNG bitmap

PNG bitmap graphic shall be transmitted through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1).

PNG bitmap graphic shall be transmitted through object carousel with the stream type value of 0X0B.

12.5.4 Coding scheme and transmission of MNG animation

For the PNG bitmap graphic data in the MNG animation format that is displayed only under the control of CLUT data specified separately from this Recommendation, the palette data within the PNG data may be omitted. MNG bitmap animation graphic shall be transmitted through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1).

12.5.5 Coding scheme and transmission of GIF graphic data and animation

GIF graphic data and animation graphics shall be transmitted through specific MPEG-2 Sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1).

12.6 Character coding and transmission - External text files referred by <media> elements

A text file encoded in ISO-8859-1 shall be transmitted through specific MPEG-2 sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1).

12.7 Transmission of XML documents

12.7.1 Transmission of NCL documents and other XML documents used in editing commands

To transmit an NCL document or another XML document file used in NCL editing command parameter, one of the following transmission methods shall be used:

a) through an interactive channel protocol;

b) through specific MPEG-2 sections.

If an interactive channel protocol is used to download an NCL Document or another XML Document file referred in a addNode editing command parameter, the uri parameter of the addDocument or addNode editing command (see Section 9) shall not have its schema equal to “x-sbtvd”, and its corresponding id parameter shall be set to NULL. The uri parameter shall specify the document location and the protocol schema used to transmit the document.

If specific MPEG-2 sections are used, several alternatives are possible, as follows. The alternative choose for SBTVD shall be in conformance with ABNT NBR 15606-3.

12.7.2 Transmission in MPEG-2 Sections

12.7.2.1 DSM-CC transport of editing commands using stream-event descriptors and object carousels

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

© ABNT 2011 - All rights reserved

Editing commands are transported in DSM-CC stream-event descriptors. DSM-CC stream-event descriptors have a very similar structure to those of event descriptors presented in Figure 5, as shown in Figure 6.

Syntax	Number of bits
StreamEventDescriptor ( ) {	
descriptorTag	8
descriptorLenght	8
eventId	16
reserved	31
eventNPT	33
privateDataLength	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 to 1928
FCS	8
}	

Figure 6 – Editing command stream event descriptor

The DSM-CC object carousel protocol allows the cyclical transmission of stream event objects and file systems. Stream event objects are used to map stream event names into stream event ids. Stream 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 themselves as listeners of stream events they handle, using event names, in this case: "*nclEditingCommand*".

Besides stream 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.

In order to transmit NCL Document files or other XML Document files, used in NCL editing command parameters, through an object carousel, the stream type value of 0x0B shall be used. In the same object carousel that carries the XML specification, a stream event object shall be transmitted in order to map the name "*nclEditingCommand*" to the eventId of the DSM-CC stream event descriptor, which shall carry an NCL editing command, as described in Clause 9.

The privateDataPayload of the stream event descriptor shall carry a set of {uri, id} reference pairs. The uri parameter of the first pair shall have the "x-sbtvd" schema and the absolute path of the XML document (the path in the data server). The corresponding id parameter in the pair shall refer to the XML Document specification IOR (carouselId, moduleId, objectKey; see ABNT NBR 15606-3 and ISO/IEC 13818-6) in the object carousel.

If other file systems has to be transmitted using other object carousels in order to complete the editing command with media content (as it is usual in the case of addDocument and addNode commands), other {uri, id} pairs shall be present in the command. In this case, the uri parameter shall have the "x-sbtvd" schema and the absolute path of file system root (the path in the datacast server), and the corresponding ior parameter in the pair shall refer to the IOR (carouselId, moduleId, objectKey; see ABNT NBR 15606-3 and ISO/IEC 13818-6) of any root child file or directory in the object carousel (the IOR of the carousel service gateway).

Figure 7 depicts an example of an NCL document transmission through an object carousel. 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 7). An object carousel shall then be generated (Service Domain = 1, in Figure 7) carrying all the interactive program contents (.ncl file and all media files) and also an event object (moduleId = 2 and objectKey = 2, in Figure 7) mapping the "*nclEditingCommand*" name to the eventId value (value "3", in Figure 7).

© ABNT 2011 - All rights reserved

A stream 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 9). The uri parameter shall have the “x-sbtvd” schema and the absolute path of the NCL document (“C:\nclRepository\weather”, in Figure 7). Finally, the IOR of the NCL document in the object carousel is carried in the xmlDocument parameter (carouselId = 1, moduleId = 1, objectKey = 2, in Figure 7).

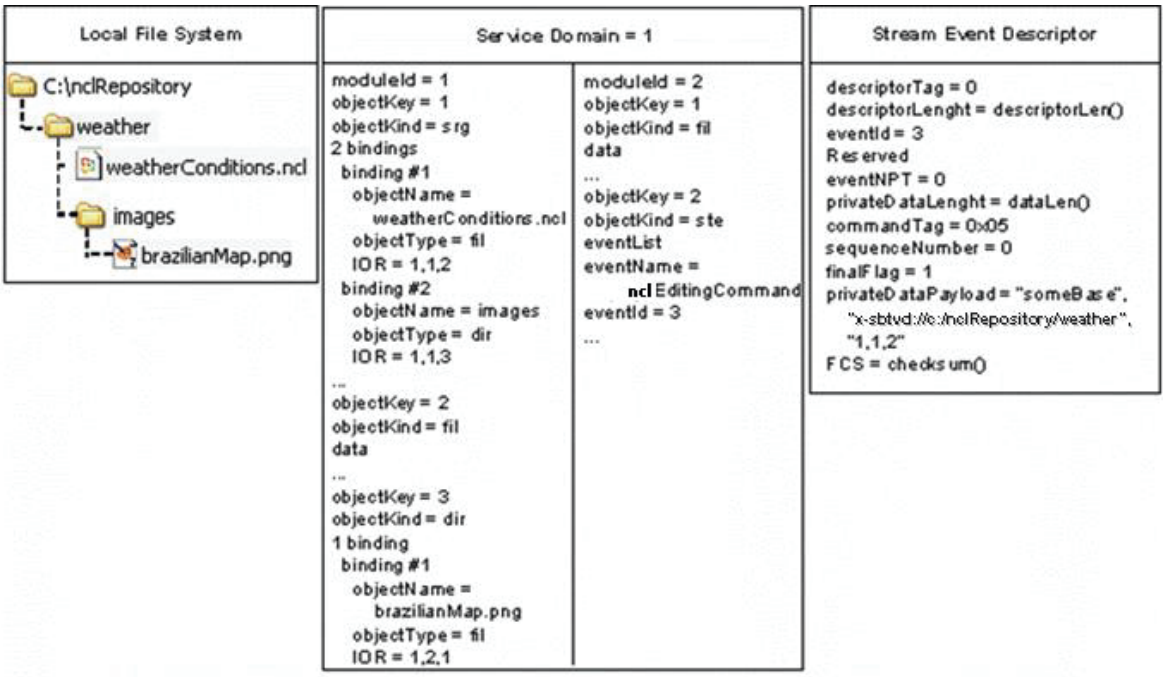

Figure 7 – Example of an NCL document transmission

12.7.2.2 Transport of editing commands using specific structures

12.7.2.2.1 Data structure definitions

Event descriptors (defined in Clause 9) can be sent in MPEG-2 TS elementary stream, using DSM-CC stream event as discussed in 12.7.1.1, or using any protocol for pushed data transmission.

Three data structure types can be 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 60. Other mappingType values may also be defined, but they are not relevant for this Standand.

Table 60 – 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
}	
}	

© ABNT 2011 - All rights reserved

Maps of type “events” (*event maps*) are used to map event names into eventIds of event descriptors (see Figure 5). 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 NCL execution-objects (for example, 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 5). One or more event descriptors with the previous selected eventId are then created and sent. 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.

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>
```


© ABNT 2011 - All rights reserved

```
        <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>

  <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.

12.7.2.2.2 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 stream.

© ABNT 2011 - All rights reserved

NOTE 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.

After the second byte comes a serialized data structure that can be a mappingStructure (as depicted by Table 60), 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.

NOTE 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 already mentioned can be wrapped in other protocol data format, like FLUTE packets.

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 Clause 9. 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 8 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 8). 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 8), mapping the "*nclEditingCommand*" name to the eventId value (value "3", in Figure 8). 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 9). The uri parameter shall have the "null" value and the id parameter shall have the (component_tag= "0x09", structureId= "0x0B", in Figure 8) value.

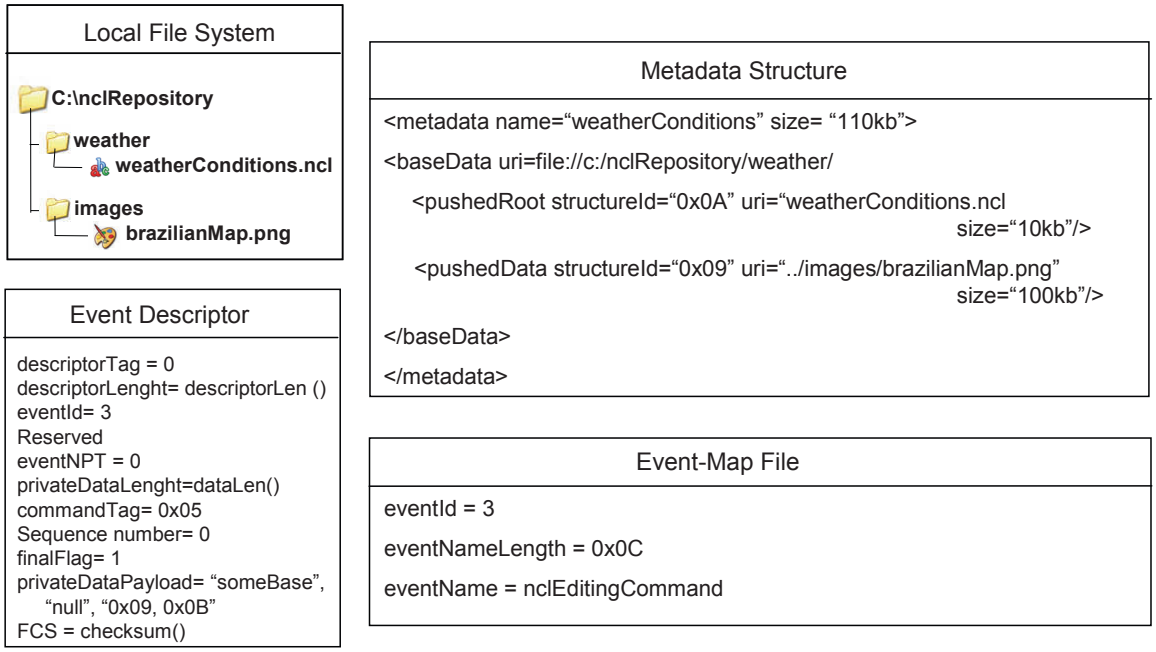

Figure 8 – Example of an NCL document transmission using MPEG-2 NCL Section

© ABNT 2011 - All rights reserved

12.7.2.2.3 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 8, 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.

12.7.2.2.4 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 61 shows the metadata section syntax for transport of metadata structures, which shall be in agreement with ISO/IEC 13818-1: 2007.

Table 61 – Section syntax for transport of metadata structures

Syntax	Number 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 62
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	
}		

© ABNT 2011 - All rights reserved

Table 62 – 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 Clause 9. 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 8, 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 9 illustrates the new situation.

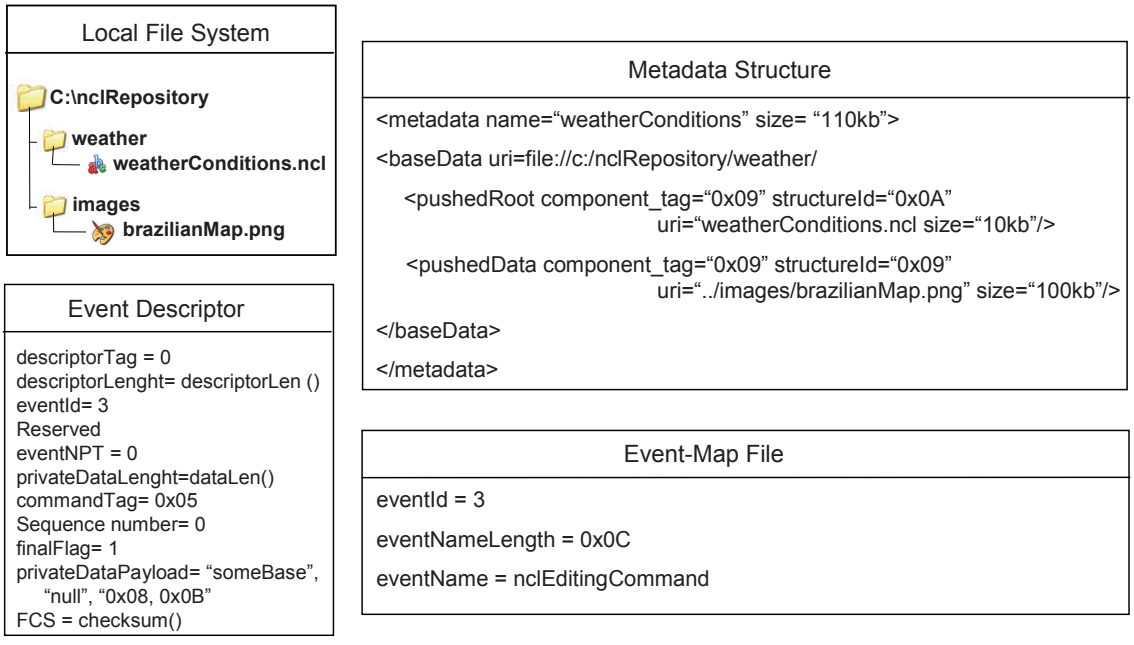

Figure 9 – Example of an NCL document transmission using MPEG-2 Metadata Section

© ABNT 2011 - All rights reserved

12.7.3 Transmission of external XML documents

External XML documents referred by <media> elements, for example, an XHTML based media object, shall be transmitted through specific MPEG-2 sections (see stream type assignments for MPEG-2 sections in ISO/IEC 13818-1).

13 Security

The Ginga security model is fully conformant to SBTVD security model, as addressed in Ginga-J specification. It addresses the same areas of security; that is, authentication of broadcast applications, security policies for applications, security over the interaction channel, and certificate management.

Authentication of Ginga-NCL applications shall be performed in the same way than for Ginga-J applications. If signed, Ginga-NCL application shall follow the signing framework as specified in Ginga-J. As such, non-authenticated Ginga-NCL applications will operate within a sandbox environment. Authenticated Ginga-NCL applications associated with a permission request file may be granted permissions outside the sandbox.

© ABNT 2011 - All rights reserved

Annex A
(normative)

NCL 3.0 module schemas used in the Basic DTV and the Enhanced DTV profiles

A.1 Structure module: NCL30Structure.xsd

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

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/modules/NCL30Structure.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the Structure module namespace,
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Structure"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- define the top-down structure of an NCL language document.          -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <complexType name="nclPrototype">
    <sequence>
      <element ref="structure:head" minOccurs="0" maxOccurs="1"/>
      <element ref="structure:body" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
  </complexType>

  <complexType name="headPrototype">
  </complexType>

  <complexType name="bodyPrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="ncl" type="structure:nclPrototype"/>
  <element name="head" type="structure:headPrototype"/>
  <element name="body" type="structure:bodyPrototype"/>

</schema>
```


© ABNT 2011 - All rights reserved

A.2 Layout module: NCL30Layout.xsd

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

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/modules/NCL30Layout.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <complexType name="regionBasePrototype">
<attribute name="id" type="ID" use="optional"/>
<attribute name="device" type="string" use="optional"/>
….<attribute name="region" type="string" use="optional"/>
  </complexType>

  <complexType name="regionPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="layout:region" />
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
    <attribute name="height" type="string" use="optional"/>
    <attribute name="left" type="string" use="optional"/>
    <attribute name="right" type="string" use="optional"/>
    <attribute name="top" type="string" use="optional"/>
    <attribute name="bottom" type="string" use="optional"/>
    <attribute name="width" type="string" use="optional"/>
    <attribute name="zIndex" type="integer" use="optional"/>
  </complexType>


  <!-- declare global attributes in this module -->

  <!-- define the region attributeGroup -->
  <attributeGroup name="regionAttrs">
    <attribute name="region" type="string" use="optional"/>
  </attributeGroup>

  <!-- declare global elements in this module -->
  <element name="regionBase" type="layout:regionBasePrototype"/>
  <element name="region" type="layout:regionPrototype"/>

</schema>
```

© ABNT 2011 - All rights reserved

A.3 Media module: NCL30Media.xsd

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

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/modules/NCL30Media.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <complexType name="mediaPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="src" type="anyURI" use="optional"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="media" type="media:mediaPrototype"/>

</schema>
```


© ABNT 2011 - All rights reserved

A.4 Context module: NCL30Context.xsd

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

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/modules/NCL30Context.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <!-- define the compositeNode element prototype -->
  <complexType name="contextPrototype">
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="context" type="context:contextPrototype"/>

</schema>
```

© ABNT 2011 - All rights reserved

A.5 MediaContentAnchor module: NCL30MediaContentAnchor.xsd

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

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/modules/NCL30MediaContentAnchor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Media Content Anchor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
 targetNamespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
 elementFormDefault="qualified" attributeFormDefault="unqualified" >

 <!-- define the temporalAnchorAttrs attribute group -->
 <attributeGroup name="temporalAnchorAttrs">
  <attribute name="begin" type="string" use="optional"/>
  <attribute name="end" type="string" use="optional"/>
 </attributeGroup>

 <!-- define the textAnchorAttrs attribute group -->
 <attributeGroup name="textAnchorAttrs">
  <attribute name="beginText" type="string" use="optional"/>
  <attribute name="beginPosition" type="unsignedLong" use="optional"/>
  <attribute name="endText" type="string" use="optional"/>
  <attribute name="endPosition" type="unsignedLong" use="optional"/>
 </attributeGroup>

 <!-- define the sampleAnchorAttrs attribute group -->
 <attributeGroup name="sampleAnchorAttrs">
  <attribute name="first" type="string" use="optional"/>
  <attribute name="last" type="string" use="optional"/>
 </attributeGroup>

 <!-- define the coordsAnchorAttrs attribute group -->
 <attributeGroup name="coordsAnchorAttrs">
  <attribute name="coords" type="string" use="optional"/>
 </attributeGroup>

 <!-- define the labelAttrs attribute group -->
 <attributeGroup name="labelAttrs">
  <attribute name="label" type="string" use="optional"/>
 </attributeGroup>

 <!-- define the clip attribute group -->
 <attributeGroup name="clipAttrs">
```


© ABNT 2011 - All rights reserved

```
    <attribute name="clip" type="string" use="optional"/>
  </attributeGroup>

  <complexType name="componentAnchorPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attributeGroup ref="mediaAnchor:coordsAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:temporalAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:textAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:sampleAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:labelAttrs" />
    <attributeGroup ref="mediaAnchor:clipAttrs" />
  </complexType>

  <!-- declare global elements in this module -->
  <element name="area" type="mediaAnchor:componentAnchorPrototype"/>

</schema>
```

© ABNT 2011 - All rights reserved

A.6 CompositeNodeInterface module: NC30CompositeNodeInterface.xsd

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

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/modules/NCL30CompositeNodeInterface.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Composite Node Interface module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  targetNamespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="compositeNodePortPrototype">
    <attribute name="id" type="ID" use="required" />
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="port" type="compositeInterface:compositeNodePortPrototype" />

</schema>
```


© ABNT 2011 - All rights reserved

A.7 PropertyAnchor module: NCL30PropertyAnchor.xsd

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

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/modules/NCL30PropertyAnchor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Property Anchor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="propertyAnchorPrototype">
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="optional" />
    <attribute name="externable" type="boolean" use="optional" />
  </complexType>


<!—
```

```
The following reserved words are used for properties' names.
* For audio media-obejcts: soundLevel; balanceLevel; trebleLevel; bassLevel.
* For text media-objects: style, which refers to a style sheet with information for text presentation; textAlign; fontColor;
fontFamily; fontStyle; fontSize; fontVariant; fontWeight.
* For visual media-objects: background, specifying the background color used to fill the area of a region displaying media; scroll,
which allows the specification of how an author would like to configure the scroll in a region; fit, indicating how an object will be
presented (hidden, fill, meet, meetBest, slice); transparency, indicating the degree of transparency of an object presentation (the
value shall be between 0 and 1, or a real value in the range [0,100] ending with the character "%" (e.g. 30%)); visible, indicating
if the presentation is to be seen or hidden; rgbChromakey; the object positioning parameters: top, left, bottom, right, width,
height, zIndex, plan, location, size and bounds; the focus movement parameters: moveLeft, moveRight, moveUp, moveDown,
focusIndex; the other related focus parameters: focusBorderColor, selBorderColor, focusBorderWidth,
focusBorderTransparency, focusSrc, and focusSelSrc; the transition parameters: transIn and transOut; the timing parameters:
explicitDur and freeze; and the multiple device parameters: baseDeviceRegion and deviceClass.
* For media-objects in general: player; reusePlayer, which determines if a new player shall be instantiated or if a player already
instantiated shall be used; and playerLife, which specifies what will happen to the player instance at the end of the presentation.
-->


  <!-- declare global elements in this module -->
```

© ABNT 2011 - All rights reserved

A.8 SwitchInterface module: NCL30SwitchInterface.xsd

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

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/modules/NCL30SwitchInterface.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Switch Interface module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  targetNamespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="mappingPrototype">
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional"/>
  </complexType>

  <complexType name="switchPortPrototype">
    <sequence>
      <element ref="switchInterface:mapping" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="mapping" type="switchInterface:mappingPrototype"/>
  <element name="switchPort" type="switchInterface:switchPortPrototype" />

</schema>
```


© ABNT 2011 - All rights reserved

A.9 Descriptor module: NCL30Descriptor.xsd

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

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/modules/NCL30Descriptor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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

  <complexType name="descriptorParamPrototype">
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="required"/>
  </complexType>

  <complexType name="descriptorPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="descriptor:descriptorParam"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="player" type="string" use="optional"/>
  </complexType>

<!--
Formatters should support the following descriptorParam names.
* For audio players: soundLevel; balanceLevel; trebleLevel; bassLevel.
* For text players: style, which refers to a style sheet with information for text presentation; textAlign; fontColor; FontFamily;
fontStyle; fontSize; fontVariant; fontWeight.
* For visual media players: background, specifying the background color used to fill the area of a region displaying media; scroll,
which allows the specification of how an author would like to configure the scroll in a region; fit, indicating how an object will be
presented (hidden, fill, meet, meetBest, slice); transparency, indicating the degree of transparency of an object presentation
(the value shall be between 0 and 1, or a real value in the range [0,100] ending with the character "%" (e.g. 30%)); visible,
indicating if the presentation is to be seen or hidden; rgbChromakey; the object positioning parameters: top, left, bottom, right,
width, height, zIndex, plan, location, size and bounds; the focus movement parameters: moveLeft, moveRight, moveUp,
moveDown, focusIndex; the other related focus parameters: focusBorderColor, selBorderColor, focusBorderWidth,
focusBorderTransparency, focusSrc, and focusSelSrc; the transition parameters: transIn and transOut; the timing parameters:
explicitDur and freeze; and the multiple device parameters: baseDeviceRegion and deviceClass.
* For players in general: player; reusePlayer, which determines if a new player shall be instantiated or if a player already
instantiated shall be used; and playerLife, which specifies what will happen to the player instance at the end of the presentation.
-->

  <complexType name="descriptorBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="descriptorParam" type="descriptor:descriptorParamPrototype"/>
  <element name="descriptor" type="descriptor:descriptorPrototype"/>
  <element name="descriptorBase" type="descriptor:descriptorBasePrototype"/>

  <!-- declare global attributes in this module -->
  <attributeGroup name="descriptorAttrs">
    <attribute name="descriptor" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

© ABNT 2011 - All rights reserved

A.10 Linking module: NCL30Linking.xsd

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

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/modules/NCL30Linking.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <complexType name="paramPrototype">
    <attribute name="name" type="string" use="required"/>
    <attribute name="value" type="anySimpleType" use="required"/>
  </complexType>

  <complexType name="bindPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="linking:bindParam"/>
    </sequence>
    <attribute name="role" type="string" use="required"/>
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional"/>
  </complexType>

  <complexType name="linkPrototype">
    <sequence>
      <element ref="linking:linkParam" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="linking:bind" minOccurs="2" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="xconnector" type="string" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="linkParam" type="linking:paramPrototype"/>
  <element name="bindParam" type="linking:paramPrototype"/>
  <element name="bind" type="linking:bindPrototype" />
  <element name="link" type="linking:linkPrototype" />

</schema>
```


© ABNT 2011 - All rights reserved

A.11 ConnectorCommonPart Module: NCL30ConnectorCommonPart.xsd

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

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/modules/NCL30ConnectorCommonPart.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Common Part module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


<complexType name="parameterPrototype">
  <attribute name="name" type="string" use="required"/>
  <attribute name="type" type="string" use="optional"/>
</complexType>

<simpleType name="eventPrototype">
  <restriction base="string">
    <enumeration value="presentation" />
    <enumeration value="selection" />
    <enumeration value="attribution" />
    <enumeration value="composition" />
  </restriction>
</simpleType>

<simpleType name="logicalOperatorPrototype">
  <restriction base="string">
    <enumeration value="and" />
    <enumeration value="or" />
  </restriction>
</simpleType>

<simpleType name="transitionPrototype">
  <restriction base="string">
    <enumeration value="starts" />
    <enumeration value="stops" />
    <enumeration value="pauses" />
    <enumeration value="resumes" />
    <enumeration value="aborts" />
  </restriction>
</simpleType>

</schema>
```

© ABNT 2011 - All rights reserved

A.12 ConnectorAssessmentExpression Module: NCL30ConnectorAssessmentExpression.xsd

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

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorAssessmentExpression.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Assessment Expression module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorAssessmentExpression="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd"/>

<simpleType name="comparatorPrototype">
  <restriction base="string">
    <enumeration value="eq" />
    <enumeration value="ne" />
    <enumeration value="gt" />
    <enumeration value="lt" />
    <enumeration value="gte" />
    <enumeration value="lte" />
  </restriction>
</simpleType>

<simpleType name="attributePrototype">
  <restriction base="string">
    <enumeration value="repeat" />
    <enumeration value="occurrences" />
    <enumeration value="state" />
    <enumeration value="nodeProperty" />
  </restriction>
</simpleType>

<simpleType name="statePrototype">
  <restriction base="string">
    <enumeration value="sleeping" />
    <enumeration value="occurring" />
    <enumeration value="paused" />
  </restriction>
</simpleType>

<simpleType name="valueUnion">
  <union memberTypes="string connectorAssessmentExpression:statePrototype"/>
</simpleType>

<complexType name="assessmentStatementPrototype" >
  <sequence>
    <element ref="connectorAssessmentExpression:attributeAssessment"/>
    <choice>
      <element ref="connectorAssessmentExpression:attributeAssessment"/>
```


© ABNT 2011 - All rights reserved

```
      <element ref="connectorAssessmentExpression:valueAssessment"/>
    </choice>
  </sequence>
  <attribute name="comparator" type="connectorAssessmentExpression:comparatorPrototype" use="required"/>
</complexType>

<complexType name="attributeAssessmentPrototype">
  <attribute name="role" type="string" use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype" use="required"/>
  <attribute name="key" type="string" use="optional"/>
  <attribute name="attributeType" type="connectorAssessmentExpression:attributePrototype" use="optional"/>
  <attribute name="offset" type="string" use="optional"/>
</complexType>

<complexType name="valueAssessmentPrototype">
  <attribute name="value" type="connectorAssessmentExpression:valueUnion" use="required"/>
</complexType>

<complexType name="compoundStatementPrototype">
  <choice minOccurs="1" maxOccurs="unbounded">
    <element ref="connectorAssessmentExpression:assessmentStatement" />
    <element ref="connectorAssessmentExpression:compoundStatement" />
  </choice>
  <attribute name="operator" type="connectorCommonPart:logicalOperatorPrototype" use="required"/>
  <attribute name="isNegated" type="boolean" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="assessmentStatement" type="connectorAssessmentExpression:assessmentStatementPrototype" />
<element name="attributeAssessment" type="connectorAssessmentExpression:attributeAssessmentPrototype" />
<element name="valueAssessment" type="connectorAssessmentExpression:valueAssessmentPrototype" />
<element name="compoundStatement" type="connectorAssessmentExpression:compoundStatementPrototype" />

</schema>
```

© ABNT 2011 - All rights reserved

A.13 ConnectorCausalExpression Module: NCL30ConnectorCausalExpression.xsd

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

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCausalExpression.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Causal Expression module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
    xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd"/>

<simpleType name="conditionRoleUnion">
  <union memberTypes="string connectorCausalExpression:conditionRolePrototype"/>
</simpleType>

<simpleType name="conditionRolePrototype">
  <restriction base="string">
    <enumeration value="onBegin" />
    <enumeration value="onEnd" />
    <enumeration value="onPause" />
    <enumeration value="onResume" />
    <enumeration value="onAbort" />
  </restriction>
</simpleType>

<simpleType name="maxUnion">
  <union memberTypes="positiveInteger connectorCausalExpression:unboundedString"/>
</simpleType>

<simpleType name="unboundedString">
  <restriction base="string">
    <pattern value="unbounded"/>
  </restriction>
</simpleType>

<complexType name="simpleConditionPrototype">
  <attribute name="role" type="connectorCausalExpression:conditionRoleUnion" use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype" use="optional"/>
  <attribute name="key" type="string" use="optional"/>
  <attribute name="transition" type="connectorCommonPart:transitionPrototype" use="optional"/>
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion" use="optional"/>
  <attribute name="qualifier" type="connectorCommonPart:logicalOperatorPrototype" use="optional"/>
</complexType>

<complexType name="compoundConditionPrototype">
  <attribute name="operator" type="connectorCommonPart:logicalOperatorPrototype" use="required"/>
  <attribute name="delay" type="string" use="optional"/>
```


© ABNT 2011 - All rights reserved

```
</complexType>

<simpleType name="actionRoleUnion">
  <union memberTypes="string connectorCausalExpression:actionNamePrototype"/>
</simpleType>

<simpleType name="actionNamePrototype">
  <restriction base="string">
    <enumeration value="start" />
    <enumeration value="stop" />
    <enumeration value="pause" />
    <enumeration value="resume" />
    <enumeration value="abort" />
    <enumeration value="set" />
  </restriction>
</simpleType>

<simpleType name="actionOperatorPrototype">
  <restriction base="string">
    <enumeration value="par" />
    <enumeration value="seq" />
  </restriction>
</simpleType>

<complexType name="simpleActionPrototype">
  <attribute name="role" type="connectorCausalExpression:actionRoleUnion" use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype" use="optional"/>
  <attribute name="actionType" type="connectorCausalExpression:actionNamePrototype" use="optional"/>
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="value" type="string" use="optional"/>
  <attribute name="repeat" type="positiveInteger" use="optional"/>
  <attribute name="repeatDelay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion" use="optional"/>
  <attribute name="qualifier" type="connectorCausalExpression:actionOperatorPrototype" use="optional"/>
</complexType>

<complexType name="compoundActionPrototype">
  <choice minOccurs="2" maxOccurs="unbounded">
    <element ref="connectorCausalExpression:simpleAction" />
    <element ref="connectorCausalExpression:compoundAction" />
  </choice>
  <attribute name="operator" type="connectorCausalExpression:actionOperatorPrototype" use="required"/>
  <attribute name="delay" type="string" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="simpleCondition" type="connectorCausalExpression:simpleConditionPrototype" />
<element name="compoundCondition" type="connectorCausalExpression:compoundConditionPrototype" />
<element name="simpleAction" type="connectorCausalExpression:simpleActionPrototype" />
<element name="compoundAction" type="connectorCausalExpression:compoundActionPrototype" />

</schema>
```

© ABNT 2011 - All rights reserved

A.14 CausalConnector module: NCL30CausalConnector.xsd

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

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30CausalConnector.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Causal Connector module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:causalConnector="http://www.ncl.org.br/NCL3.0/CausalConnector"
  targetNamespace="http://www.ncl.org.br/NCL3.0/CausalConnector"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="causalConnectorPrototype">
  <attribute name="id" type="ID" use="required"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="causalConnector" type="causalConnector:causalConnectorPrototype"/>
</schema>
```


© ABNT 2011 - All rights reserved

A.15 ConnectorBase module: NCL30ConnectorBase.xsd

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

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorBase.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Base module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="connectorBasePrototype">
  <attribute name="id" type="ID" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="connectorBase" type="connectorBase:connectorBasePrototype"/>
</schema>
```

© ABNT 2011 - All rights reserved

A.16 NCL30CausalConnectorFunctionality.xsd

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

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

Public URI: http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnectorFunctionality.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL CausalConnectorFunctionality module namespace.
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/
ConnectorCommonPart"
        xmlns:connectorAssessmentExpression="http://www.ncl.org.br/NCL3.0/
ConnectorAssessmentExpression"
        xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/
ConnectorCausalExpression"
        xmlns:causalConnector="http://www.ncl.org.br/NCL3.0/
CausalConnector"
        xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
        targetNamespace="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
        elementFormDefault="qualified" attributeFormDefault="unqualified">

   <!-- import the definitions in the modules namespaces -->

   <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCommonPart.xsd"/>
   <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorAssessmentExpression.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/CausalConnector"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnector.xsd"/>

  <!-- =================================================================== -->
  <!-- CausalConnectorFunctionality                               -->
  <!-- =================================================================== -->
  <element name="connectorParam" type="connectorCommonPart:parameterPrototype"/>

   <!-- extends causalConnector element -->

   <complexType name="causalConnectorType">
     <complexContent>
       <extension base="causalConnector:causalConnectorPrototype">
         <sequence>
           <element ref="causalConnectorFunctionality:connectorParam" minOccurs="0" maxOccurs="unbounded"/>
           <choice>
             <element ref="causalConnectorFunctionality:simpleCondition" />
             <element ref="causalConnectorFunctionality:compoundCondition" />
           </choice>
```


© ABNT 2011 - All rights reserved

```
          <choice>
            <element ref="causalConnectorFunctionality:simpleAction" />
            <element ref="causalConnectorFunctionality:compoundAction" />
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- extends compoundCondition element -->

  <complexType name="compoundConditionType">
    <complexContent>
      <extension base="connectorCausalExpression:compoundConditionPrototype">
        <sequence>
          <choice>
            <element ref="causalConnectorFunctionality:simpleCondition" />
            <element ref="causalConnectorFunctionality:compoundCondition" />
          </choice>
          <choice minOccurs="1" maxOccurs="unbounded">
            <element ref="causalConnectorFunctionality:simpleCondition" />
            <element ref="causalConnectorFunctionality:compoundCondition" />
            <element ref="causalConnectorFunctionality:assessmentStatement" />
            <element ref="causalConnectorFunctionality:compoundStatement" />
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

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

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

  <element name="compoundCondition" type="causalConnectorFunctionality:compoundConditionType"
substitutionGroup="connectorCausalExpression:compoundCondition"/>

  <element name="simpleAction" substitutionGroup="connectorCausalExpression:simpleAction"/>

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

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

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

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

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

</schema>
```

© ABNT 2011 - All rights reserved

A.17 TestRule module: NCL30TestRule.xsd

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

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/modules/NCL30TestRule.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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

  <complexType name="rulePrototype">
<attribute name="id" type="ID" use="optional"/>
    <attribute name="var" type="string" use="required"/>
    <attribute name="value" type="string" use="required"/>
    <attribute name="comparator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="eq"/>
          <enumeration value="ne"/>
          <enumeration value="gt"/>
          <enumeration value="gte"/>
          <enumeration value="lt"/>
          <enumeration value="lte"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="compositeRulePrototype">
    <choice minOccurs="2" maxOccurs="unbounded">
      <element ref="testRule:rule"/>
      <element ref="testRule:compositeRule"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="operator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="and"/>
          <enumeration value="or"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="ruleBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="rule" type="testRule:rulePrototype"/>
  <element name="compositeRule" type="testRule:compositeRulePrototype"/>
  <element name="ruleBase" type="testRule:ruleBasePrototype"/>

</schema>
```


© ABNT 2011 - All rights reserved

A.18 TestRuleUse module: NCL30TestRuleUse.xsd

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

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/modules/NCL30TestRuleUse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <complexType name="bindRulePrototype">
    <attribute name="constituent" type="IDREF" use="required" />
    <attribute name="rule" type="string" use="required" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="bindRule" type="testRule:bindRulePrototype"/>

</schema>
```

© ABNT 2011 - All rights reserved

A.19 ContentControl module: NCL30ContentControl.xsd

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

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

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ContentControl.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <complexType name="defaultComponentPrototype">
    <attribute name="component" type="IDREF" use="required" />
  </complexType>

  <!-- define the switch element prototype -->

  <complexType name="switchPrototype">
    <choice>
    <element ref="contentControl:defaultComponent" minOccurs="0" maxOccurs="1"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="defaultComponent" type="contentControl:defaultComponentPrototype"/>
  <element name="switch" type="contentControl:switchPrototype"/>

</schema>
```


© ABNT 2011 - All rights reserved

A.20 DescriptorControl module: NCL30DescriptorControl.xsd

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

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

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30DescriptorControl.xsd
Author: TeleMidia Laboratory
Revision: 19/06/2006

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


  <complexType name="defaultDescriptorPrototype">
    <attribute name="descriptor" type="IDREF" use="required" />
  </complexType>

  <!-- define the descriptor switch element prototype -->
  <complexType name="descriptorSwitchPrototype">
    <choice>
    <element ref="descriptorControl:defaultDescriptor" minOccurs="0" maxOccurs="1"/>
    </choice>
    <attribute name="id" type="ID" use="required”/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="defaultDescriptor" type="descriptorControl:defaultDescriptorPrototype"/>
  <element name="descriptorSwitch" type="descriptorControl:descriptorSwitchPrototype"/>

</schema>
```

© ABNT 2011 - All rights reserved

A.21 Timing module: NCL30Timing.xsd

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

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/modules/NCL30Timing.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <!-- declare global attributes in this module -->

  <!-- define the explicitDur attribute group -->
  <attributeGroup name="explicitDurAttrs">
    <attribute name="explicitDur" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the freeze attribute group -->
  <attributeGroup name="freezeAttrs">
    <attribute name="freeze" type="boolean" use="optional"/>
  </attributeGroup>

</schema>
```


© ABNT 2011 - All rights reserved

A.22 Import module: NCL30Import.xsd

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

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/modules/NCL30Import.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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

<complexType name="importBasePrototype">
<attribute name="alias" type="ID" use="required"/>
<attribute name="region" type="IDREF" use="optional"/>
<attribute name="documentURI" type="anyURI" use="required"/>
<attribute name="baseId" type="IDREF" use="optional"/>
</complexType>

  <complexType name="importNCLPrototype">
     <attribute name="alias" type="ID" use="required"/>
     <attribute name="documentURI" type="anyURI" use="required"/>
  </complexType>

  <complexType name="importedDocumentBasePrototype">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element ref="import:importNCL" />
    </sequence>
    <attribute name="id" type="ID" use="optional" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="importBase" type="import:importBasePrototype"/>
  <element name="importNCL" type="import:importNCLPrototype"/>
  <element name="importedDocumentBase" type="import:importedDocumentBasePrototype"/>

</schema>
```

© ABNT 2011 - All rights reserved

A.23 EntityReuse module: NCL30EntityReuse.xsd

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

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/modules/NCL30EntityReuse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <attributeGroup name="entityReuseAttrs">
    <attribute name="refer" type="string" use="optional"/>
  </attributeGroup>

</schema>
```


© ABNT 2011 - All rights reserved

A.24 ExtendedEntityReuse module: NCL30ExtendedEntityReuse.xsd

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

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/modules/NCL30ExtendedEntityReuse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <attributeGroup name="extendedEntityReuseAttrs">
    <attribute name="instance" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

© ABNT 2011 - All rights reserved

A.25 KeyNavigation module: NCL30KeyNavigation.xsd

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

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/modules/NCL30KeyNavigation.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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

<simpleType name="colorPrototype">
  <restriction base="string">
    <enumeration value="white" />
    <enumeration value="black" />
    <enumeration value="silver" />
    <enumeration value="gray" />
    <enumeration value="red" />
    <enumeration value="maroon" />
    <enumeration value="fuchsia" />
    <enumeration value="purple" />
    <enumeration value="lime" />
    <enumeration value="green" />
    <enumeration value="yellow" />
    <enumeration value="olive" />
    <enumeration value="blue" />
    <enumeration value="navy" />
    <enumeration value="aqua" />
    <enumeration value="teal" />
  </restriction>
</simpleType>

  <!-- declare global attributes in this module -->

  <!-- define the keyNavigation attribute group -->
  <attributeGroup name="keyNavigationAttrs">
    <attribute name="moveLeft" type="positiveInteger" use="optional"/>
    <attribute name="moveRight" type="positiveInteger" use="optional"/>
    <attribute name="moveUp" type="positiveInteger" use="optional"/>
    <attribute name="moveDown" type="positiveInteger" use="optional"/>
    <attribute name="focusIndex" type="positiveInteger" use="optional"/>
    <attribute name="focusBorderColor" type="keyNavigation:colorPrototype" use="optional"/>
    <attribute name="focusBorderWidth" type="string" use="optional"/>
    <attribute name="focusBorderTransparency" type="string" use="optional"/>
    <attribute name="focusSrc" type="string" use="optional"/>
    <attribute name="focusSelSrc" type="string" use="optional"/>
    <attribute name="selBorderColor" type="keyNavigation:colorPrototype" use="optional"/>
  </attributeGroup>


</schema>
```


© ABNT 2011 - All rights reserved

A.26 TransitionBase module: NCL30TransitionBase.xsd

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

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30TransitionBase.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Transition Base module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:transitionBase="http://www.ncl.org.br/NCL3.0/TransitionBase"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TransitionBase"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="transitionBasePrototype">
  <attribute name="id" type="ID" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="transitionBase" type="transitionBase:transitionBasePrototype"/>
</schema>
```

© ABNT 2011 - All rights reserved

A.27 Animation module: NCL30Animation.xsd

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

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/modules/NCL30Animation.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Timing module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:animation="http://www.ncl.org.br/NCL3.0/Animation"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Animation"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- declare global attributes in this module -->

  <!-- define the animation attribute group -->
  <attributeGroup name="animationAttrs">
    <attribute name="duration" type="string" use="optional"/>
    <attribute name="by" type="string" use="optional"/>
  </attributeGroup>

</schema>
```


© ABNT 2011 - All rights reserved

A.28 Transition module: NCL30Transition.xsd

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

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Transition.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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


  <!-- declare global attributes in this module -->

  <!-- define the type attribute prototype -->
  <simpleType name="typePrototype">
    <restriction base="string">
      <enumeration value="in"/>
      <enumeration value="barWipe"/>
      <enumeration value="boxWipe"/>
      <enumeration value="fourBoxWipe"/>
      <enumeration value="barnDoorWipe"/>
      <enumeration value="diagonalWipe"/>
      <enumeration value="bowTieWipe"/>
      <enumeration value="miscDiagonalWipe"/>
      <enumeration value="veeWipe"/>
      <enumeration value="barnVeeWipe"/>
      <enumeration value="zigZagWipe"/>
      <enumeration value="barnZigZagWipe"/>
      <enumeration value="irisWipe"/>
      <enumeration value="triangleWipe"/>
      <enumeration value="arrowHeadWipe"/>
      <enumeration value="pentagonWipe"/>
      <enumeration value="hexagonWipe"/>
      <enumeration value="ellipseWipe"/>
      <enumeration value="eyeWipe"/>
      <enumeration value="roundRectWipe"/>
      <enumeration value="starWipe"/>
      <enumeration value="miscShapeWipe"/>
      <enumeration value="clockWipe"/>
      <enumeration value="pinWheelWipe"/>
      <enumeration value="singleSweepWipe"/>
      <enumeration value="fanWipe"/>
      <enumeration value="doubleFanWipe"/>
      <enumeration value="doubleSweepWipe"/>
      <enumeration value="saloonDoorWipe"/>
      <enumeration value="windshieldWipe"/>
      <enumeration value="snakeWipe"/>
      <enumeration value="spiralWipe"/>
      <enumeration value="parallelSnakesWipe"/>
      <enumeration value="boxSnakesWipe"/>
      <enumeration value="waterfallWipe"/>
      <enumeration value="pushWipe"/>
      <enumeration value="slideWipe"/>
```

© ABNT 2011 - All rights reserved

```
      <enumeration value="fade"/>
      <enumeration value="audioFade"/>
      <enumeration value="audioVisualFade"/>
    </restriction>
  </simpleType>

  <!-- define subType attribute prototype-->
  <simpleType name="subTypePrototype">
    <restriction base="string">
      <enumeration value="bottom"/>
      <enumeration value="bottomCenter"/>
      <enumeration value="bottomLeft"/>
      <enumeration value="bottomLeftClockwise"/>
      <enumeration value="bottomLeftCounterClockwise"/>
      <enumeration value="bottomLeftDiagonal"/>
      <enumeration value="bottomRight"/>
      <enumeration value="bottomRightClockwise"/>
      <enumeration value="bottomRightCounterClockwise"/>
      <enumeration value="bottomRightDiagonal"/>
      <enumeration value="centerRight"/>
      <enumeration value="centerTop"/>
      <enumeration value="circle"/>
      <enumeration value="clockwiseBottom"/>
      <enumeration value="clockwiseBottomRight"/>
      <enumeration value="clockwiseLeft"/>
      <enumeration value="clockwiseNine"/>
      <enumeration value="clockwiseRight"/>
      <enumeration value="clockwiseSix"/>
      <enumeration value="clockwiseThree"/>
      <enumeration value="clockwiseTop"/>
      <enumeration value="clockwiseTopLeft"/>
      <enumeration value="clockwiseTwelve"/>
      <enumeration value="cornersIn"/>
      <enumeration value="cornersOut"/>
      <enumeration value="counterClockwiseBottomLeft"/>
      <enumeration value="counterClockwiseTopRight"/>
      <enumeration value="crossfade"/>
      <enumeration value="diagonalBottomLeft"/>
      <enumeration value="diagonalBottomLeftOpposite"/>
      <enumeration value="diagonalTopLeft"/>
      <enumeration value="diagonalTopLeftOpposite"/>
      <enumeration value="diamond"/>
      <enumeration value="doubleBarnDoor"/>
      <enumeration value="doubleDiamond"/>
      <enumeration value="down"/>
      <enumeration value="fadeFromColor"/>
      <enumeration value="fadeToColor"/>
      <enumeration value="fanInHorizontal"/>
      <enumeration value="fanInVertical"/>
      <enumeration value="fanOutHorizontal"/>
      <enumeration value="fanOutVertical"/>
      <enumeration value="fivePoint"/>
      <enumeration value="fourBlade"/>
      <enumeration value="fourBoxHorizontal"/>
      <enumeration value="fourBoxVertical"/>
      <enumeration value="fourPoint"/>
      <enumeration value="fromBottom"/>
      <enumeration value="fromLeft"/>
      <enumeration value="fromRight"/>
      <enumeration value="fromTop"/>
      <enumeration value="heart"/>
      <enumeration value="horizontal"/>
      <enumeration value="horizontalLeft"/>
      <enumeration value="horizontalLeftSame"/>
      <enumeration value="horizontalRight"/>
```


© ABNT 2011 - All rights reserved

```
      <enumeration value="horizontalRightSame"/>
      <enumeration value="horizontalTopLeftOpposite"/>
      <enumeration value="horizontalTopRightOpposite"/>
      <enumeration value="keyhole"/>
      <enumeration value="left"/>
      <enumeration value="leftCenter"/>
      <enumeration value="leftToRight"/>
      <enumeration value="oppositeHorizontal"/>
      <enumeration value="oppositeVertical"/>
      <enumeration value="parallelDiagonal"/>
      <enumeration value="parallelDiagonalBottomLeft"/>
      <enumeration value="parallelDiagonalTopLeft"/>
      <enumeration value="parallelVertical"/>
      <enumeration value="rectangle"/>
      <enumeration value="right"/>
      <enumeration value="rightCenter"/>
      <enumeration value="sixPoint"/>
      <enumeration value="top"/>
      <enumeration value="topCenter"/>
      <enumeration value="topLeft"/>
      <enumeration value="topLeftClockwise"/>
      <enumeration value="topLeftCounterClockwise"/>
      <enumeration value="topLeftDiagonal"/>
      <enumeration value="topLeftHorizontal"/>
      <enumeration value="topLeftVertical"/>
      <enumeration value="topRight"/>
      <enumeration value="topRightClockwise"/>
      <enumeration value="topRightCounterClockwise"/>
      <enumeration value="topRightDiagonal"/>
      <enumeration value="topToBottom"/>
      <enumeration value="twoBladeHorizontal"/>
      <enumeration value="twoBladeVertical"/>
      <enumeration value="twoBoxBottom"/>
      <enumeration value="twoBoxLeft"/>
      <enumeration value="twoBoxRight"/>
      <enumeration value="twoBoxTop"/>
      <enumeration value="up"/>
      <enumeration value="vertical"/>
      <enumeration value="verticalBottomLeftOpposite"/>
      <enumeration value="verticalBottomSame"/>
      <enumeration value="verticalLeft"/>
      <enumeration value="verticalRight"/>
      <enumeration value="verticalTopLeftOpposite"/>
      <enumeration value="verticalTopSame"/>
    </restriction>
  </simpleType>

  <attributeGroup name="transAttrs">
    <attribute name="transIn" type="string" use="optional"/>
    <attribute name="transOut" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the transition attribute group -->
  <attributeGroup name="transitionAttrs">
    <attribute name="type" type="transition:typePrototype" use="required"/>
    <attribute name="subtype" type="transition:subTypePrototype" use="optional"/>
    <attribute name="fadeColor" type="string" use="optional" default="black"/>
    <attribute name="dur" type="string" use="optional"/>
    <attribute name="startProgress" use="optional" default="0.0">
      <simpleType>
        <restriction base="decimal">
          <minInclusive value="0.0"/>
          <maxInclusive value="1.0"/>
        </restriction>
      </simpleType>
```

© ABNT 2011 - All rights reserved

```
    </attribute>
    <attribute name="endProgress" use="optional" default="1.0">
      <simpleType>
        <restriction base="decimal">
          <minInclusive value="0.0"/>
          <maxInclusive value="1.0"/>
        </restriction>
      </simpleType>
    </attribute>
    <attribute name="direction" use="optional" default="forward">
      <simpleType>
        <restriction base="string">
          <enumeration value="forward"/>
          <enumeration value="reverse"/>
        </restriction>
      </simpleType>
    </attribute>
  </attributeGroup>

  <!-- define the transition-modifier attribute group -->
  <attributeGroup name="transitionModifierAttrs">
    <attribute name="horzRepeat" type="decimal" use="optional"  default="1.0"/>
    <attribute name="vertRepeat" type="decimal" use="optional"  default="1.0"/>
    <attribute name="borderWidth" type="nonNegativeInteger" use="optional"  default="0"/>
    <attribute name="borderColor" type="string" use="optional"  default="black"/>
  </attributeGroup>

  <complexType name="transitionPrototype">
  <attribute name="id" type="ID" use="required"/>
     <attributeGroup ref="transition:transitionAttrs"/>
     <attributeGroup ref="transition:transitionModifierAttrs"/>
  </complexType>

  <!-- declare global element in this module -->
  <element name="transition" type="transition:transitionPrototype"/>

</schema>
```


© ABNT 2011 - All rights reserved

A.29 Metainformation module: NCL30Metainformation.xsd

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

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/modules/NCL30Metainformation.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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

  <complexType name="metaPrototype">
    <attribute name="name" type="string" use="required"/>
    <attribute name="content" type="string" use="required"/>
  </complexType>

  <complexType name="metadataPrototype">
     <sequence>
          <any minOccurs="0"/>
     </sequence>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="meta" type="metainformation:metaPrototype"/>

  <!-- declare global elements in this module -->
  <element name="metadata" type="metainformation:metadataPrototype"/>

</schema>
```

© ABNT 2011 - All rights reserved

Anexo B
(informativo)

Lua 5.1 reference manual

B.1 Introduction

NOTE This Annex shows the specification of Lua 5.1. This content é a translation of Reference Manual, by Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes, Lua.org, August 2006 (ISBN 85-903798-3-3), and it is reprinted here with approval of the authors.

Lua is an extension programming language designed to support general imperative programming with data description facilities. It also offers support for object-oriented programming, functional programming, and data-driven programming. Lua is intended to be used as a powerful, lightweight scripting language for any program that needs one. Lua is implemented as a library, written in *clean* C (that is, in the common subset of ANSI C and C++).

Being an extension language, Lua has no notion of a "main" program: it only works *embedded* in a host client, called the *embedding program* or simply the *host*. This host program may invoke functions to execute a piece of Lua code, may write and read Lua variables, and may register C functions to be called by Lua code. Through the use of C functions, Lua may be augmented to cope with a wide range of different domains, thus creating customized programming languages sharing a syntactical framework. The Lua distribution includes a sample host program called `lua`, which uses the Lua library to offer a complete, stand-alone Lua interpreter.

Lua is free software, and is provided as usual with no guarantees, as stated in its license. The implementation described in this manual, as weel as some technical papers, are available at Lua's official web site, http://www.lua.org.

B.2 The language

B.2.1 Used notation

The language constructs are explained using the usual extended BNF notation, in which {*a*} means 0 or more *a*'s, and [*a*] means an optional *a*. Keywords are shown in **bold**, non-terminals are shown in the standard document font, and other terminal symbols are shown in ‘=”. The complete syntax of Lua is described in B.8.

B.2.2 Lexical conventions

Names (also called *identifiers*) in Lua may be any string of letters, digits, and underscores, not beginning with a digit. This coincides with the definition of names in most languages. (The definition of letter depends on the current locale: any character considered alphabetic by the current locale may be used in an identifier.) Identifiers are used to name variables and table fields.

The following *keywords* are reserved and shall not be used as names:

```
and       break     do      else      elseif
end       false     for     function   if
in        local     nil     not       or
repeat    return    then    true      until   while
```


© ABNT 2011 - All rights reserved

Lua is a case-sensitive language: and is a reserved word, but And and AND are two different, valid names. As a convention, names starting with an underscore followed by uppercase letters (such as `_VERSION`) are reserved for internal global variables used by Lua.

The following strings denote other tokens:

```
+     -     *     /     %     ^     #
==    ~=    <=    >=    <     >     =
(     )     {     }     [     ]
;     :     ,     .     ..    ...
```

Literal strings may be delimited by matching single or double quotes, and may contain the following C-like escape sequences: '\a' (bell), '\b' (backspace), '\f' (form feed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\v' (vertical tab), '\\' (backslash), '\"' (quotation mark [double quote]) and '\'' (apostrophe [single quote]). Moreover, a backslash followed by a real newline results in a newline in the string. A character in a string may also be specified by its numerical value using the escape sequence `\*ddd*´, where *ddd* is a sequence of up to three decimal digits. (Note that if a numerical escape is to be followed by a digit, it must be expressed using exactly three digits.) Strings in Lua may contain any 8-bit value, including embedded zeros, which may be specified as `\0´.

To put a double (single) quote, a newline, a backslash, or an embedded zero inside a literal string enclosed by double (single) quotes you must use an escape sequence. Any other character may be directly inserted into the literal. (Some control characters can cause problems for the file system, but Lua has no problem with them.)

Literal strings may also be defined using a long format enclosed by *long brackets*. We define an *opening long bracket of level n* as an opening square bracket followed by *n* equal signs followed by another opening square bracket. So, an opening long bracket of level 0 is written as `[[`, an opening long bracket of level 1 is written as `[=[`, and so on. A *closing long bracket* is defined similarly; for instance, a closing long bracket of level 4 is written as `]====]`. A long string starts with an opening long bracket of any level and ends at the first closing long bracket of the same level. Literals in this bracketed form can run for several lines, do not interpret any escape sequences, and ignore long brackets of any other level. They may contain anything except a closing bracket of the proper level or embedded zeros.

For convenience, when the opening long bracket is immediately followed by a newline, the newline is not included in the string. As an example, in a system using ASCII (in which `a´ is coded as 97, newline is coded as 10, and `1´ is coded as 49), the four literals below denote the same string:

```
a = 'alo\n123"'
    a = "alo\n123\""
    a = '\97lo\10\04923"'
    a = [[alo
    123"]]
    a = [==[
    alo
    123"]==]
```

Numerical constants may be written with an optional decimal part and an optional decimal exponent. Examples of valid numerical constants are

```
3   3.0   3.1416   314.16e-2   0.31416E1   0xff   0x56
```

Comments start with a double hyphen (`--`) anywhere outside a string. If the text immediately after `--` is not an opening long bracket, the comment is a *short comment*, which runs until the end of the line. Otherwise, it is a *long comment*, which runs until the corresponding closing long bracket. Long comments are frequently used to disable code temporarily.

© ABNT 2011 - All rights reserved

B.2.3 Values and types

B.2.3.1 Basic types

Lua is a *dynamically* typed language. That means that variables do not have types; only values do. There are no type definitions in the language. All values carry their own type.

All values in Lua are *first-class values*. That means that all values may be stored in variables, passed as arguments to other functions, and returned as results.

There are eight basic types in Lua: *nil*, *boolean*, *number*, *string*, *function*, *userdata*, *thread*, and *table*. *Nil* is the type of the value **nil**, whose main property is to be different from any other value; it usually represents the absence of a useful value. *Boolean* is the type of the values **false** and **true**. Both **nil** and **false** make a condition false; any other value makes it true. *Number* represents real (double-precision floating-point) numbers. It is easy to build Lua interpreters that use other internal representations for numbers, such as single-precision float or long integers. See file `luaconf.h`.) *String* represents arrays of characters. Lua is 8-bit clean: Strings may contain any 8-bit character, including embedded zeros (`\0´) (see B.2.2).

Lua may call (and manipulate) functions written in Lua and functions written in C (see B.2.6.9).

The type *userdata* is provided to allow arbitrary C data to be stored in Lua variables. This type corresponds to a block of raw memory and has no pre-defined operations in Lua, except assignment and identity test. However, by using *metatables*, the programmer may define operations for userdata values (see B.2.9). Userdata values shall not be created or modified in Lua, only through the C API. This guarantees the integrity of data owned by the host program.

The type *thread* represents independent threads of execution and it is used to implement coroutines (see B.2.12). Do not confuse Lua threads with operating-system threads. Lua supports coroutines on all systems, even those that do not support threads.

The type *table* implements associative arrays, that is, arrays that may be indexed not only with numbers, but with any value (except **nil**). Tables may be *heterogeneous*; that is, they may contain values of all types (except **nil**). Tables are the sole data structuring mechanism in Lua; they may be used to represent ordinary arrays, symbol tables, sets, records, graphs, trees, etc. To represent records, Lua uses the field name as an index. The language supports this representation by providing `a.name` as syntactic sugar for `a["name"]`. There are several convenient ways to create tables in Lua (see B.2.6.8).

Like indices, the value of a table field may be of any type (except **nil**). In particular, because functions are first-class values, table fields may contain functions. Thus tables may also carry *methods* (see B.2.6.10).

Strings, tables, functions, and userdata values are *objects*: variables do not actually *contain* these values, only *references* to them. Assignment, parameter passing, and function returns always manipulate references to such values; these operations do not imply any kind of copy.

The library function `type` returns a string describing the type of a given value.

B.2.3.2 Coercion

Lua provides automatic conversion between string and number values at run time. Any arithmetic operation applied to a string tries to convert that string to a number, following the usual conversion rules. Conversely, whenever a number is used where a string is expected, the number is converted to a string, in a reasonable format. For complete control over how numbers are converted to strings, use the `format` function from the string library (see string.format).

© ABNT 2011 - All rights reserved

B.2.4 Variables

Variables are places that store values.

There are three kinds of variables in Lua: global variables, local variables, and table fields.

A single name may denote a global variable or a local variable (or a function formal parameter, which is a particular kind of local variable):

```
var ::= Name
```

Name denotes identifiers, as defined in (see B.2.2).

Variables are assumed to be global unless explicitly declared local (see B.2.5.8). Local variables are *lexically scoped*: Local variables can be freely accessed by functions defined inside their scope (see B.2.7).

Before the first assignment to a variable, its value is **nil**.

Square brackets are used to index a table:

```
var ::= prefixexp `[´ exp `]´
```

The meaning of accesses to global variables and table fields may be changed via metatables. An access to an indexed variable t[i] is equivalent to a call gettable_event(t,i) (see B.2.9 for a complete description of the gettable_event function. This function is not defined or callable in Lua. We use it here only for explanatory purposes).

The syntax `var.Name` is just syntactic sugar for `var["Name"]`:

```
var ::= prefixexp `.´ Name
```

All global variables live as fields in ordinary Lua tables, called *environment tables* or simply *environments* (see B.2.10). Each function has its own reference to an environment, so that all global variables in that function will refer to that environment table. When a function is created, it inherits the environment from the function that created it. To get the environment table of a Lua function, you call `getfenv`. To replace it, shall calll `setfenv` (only way to manipulate the environment of C functions is through the debug library; see B.5.11).

An access to a global variable x is equivalent to _env.x, which in turn is equivalent to

```
gettable_event(_env, "x")
```

where _env is the environment of the running function (see B.2.9 for a complete description of the gettable_event function. This function is not defined or callable in Lua. Similarly, the _env variable is not defined in Lua. They were used them here only for explanatory purposes).

B.2.5 Statements

B.2.5.1 Basic concepts

Lua supports an almost conventional set of statements, similar to those in Pascal or C. This set includes assignment, control structures, function calls, table constructors, and variable declarations.

© ABNT 2011 - All rights reserved

B.2.5.2 Chunks

The unit of execution of Lua is called a *chunk*. A chunk is simply a sequence of statements, which are executed sequentially. Each statement may be optionally followed by a semicolon:

```
chunk ::= {stat [`;´]}
```

There are no empty statements and thus `;;´ is not legal.

Lua handles a chunk as the body of an anonymous function with a variable number of arguments (see B.2.6.10). As such, chunks may define local variables, receive arguments, and return values.

A chunk may be stored in a file or in a string inside the host program. When a chunk is executed, first it is pre-compiled into instructions for a virtual machine, and then the compiled code is executed by an interpreter for the virtual machine.

Chunks may also be pre-compiled into binary form; see program `luac` for details. Programs in source and compiled forms are interchangeable; Lua automatically detects the file type and acts accordingly.

B.2.5.3 Blocks

A block is a list of statements; syntactically, a block is the same as a chunk:

```
block ::= chunk
```

A block may be explicitly delimited to produce a single statement:

```
stat ::= do block end
```

Explicit blocks are useful to control the scope of variable declarations. Explicit blocks are also sometimes used to add a **return** or **break** statement in the middle of another block (see B.2.5.5).

B.2.5.4 Assignment

Lua allows multiple assignment. Therefore, the syntax for assignment defines a list of variables on the left side and a list of expressions on the right side. The elements in both lists are separated by commas:

```
stat ::= varlist1 `=´ explist1

varlist1 ::= var {`,´ var}

explist1 ::= exp {`,´ exp}
```

Expressions are discussed in B.2.6.

Before the assignment, the list of values is *adjusted* to the length of the list of variables. If there are more values than needed, the excess values are thrown away. If there are fewer values than needed, the list is extended with as many **nil**'s as needed. If the list of expressions ends with a function call, then all values returned by that call enter in the list of values, before the adjustment (except when the call is enclosed in parentheses; see B.2.6).

The assignment statement first evaluates all its expressions and only then are the assignments performed. Thus the code

```
i = 3

i, a[i] = i+1, 20
```


© ABNT 2011 - All rights reserved

sets a[3] to 20, without affecting a[4] because the i in a[i] is evaluated (to 3) before it is assigned 4. Similarly, the line

```
x, y = y, x
```

exchanges the values of x and y.

The meaning of assignments to global variables and table fields may be changed via metatables. An assignment to an indexed variable t[i] = val is equivalent to settable_event(t,i,val) (see B.2.9 for a complete description of the settable_event function. This function is not defined or callable in Lua. We use it here only for explanatory purposes.)

An assignment to a global variable x = val is equivalent to the assignment _env.x = val, which in turn is equivalent to

```
settable_event(_env, "x", val)
```

where `_env` is the environment of the running function (the `_env` variable is not defined in Lua. They were used it here only for explanatory purposes).

B.2.5.5 Control structures

The control structures **if**, **while**, and **repeat** have the usual meaning and familiar syntax:

```
stat ::= while exp do block end

stat ::= repeat block until exp

stat ::= if exp then block {elseif exp then block} [else block] end
```

Lua also has a **for** statement, in two flavors (see B.2.5.6).

The condition expression of a control structure may return any value. Both **false** and **nil** are considered false. All values different from **nil** and **false** are considered true (in particular, the number 0 and the empty string are also true).

In the **repeat**--**until** loop, the inner block does not end at the **until** keyword, but only after the condition. So, the condition may refer to local variables declared inside the loop block.

The **return** statement is used to return values from a function or a chunk (which is just a function). Functions and chunks may return more than one value, so the syntax for the **return** statement is

```
stat ::= return [explist1]
```

The **break** statement is used to terminate the execution of a **while**, **repeat**, or for loop, skipping to the next statement after the loop:

```
stat ::= break
```

A **break** ends the innermost enclosing loop.

The **return** and **break** statements may only be written as the *last* statement of a block. If it is really necessary to **return** or **break** in the middle of a block, then an explicit inner block may be used, as in the idioms `` `do return end´ `` and `` `do break end´ ``, because now **return** and **break** are the last statements in their (inner) blocks.

© ABNT 2011 - All rights reserved

B.2.5.6 For statement

The **for** statement has two forms: one numeric and one generic.

The numeric **for** loop repeats a block of code while a control variable runs through an arithmetic progression. It has the following syntax:

```
stat ::= for name `=´ exp `,´ exp [`,´ exp] do block end
```

The *block* is repeated for *name* starting at the value of the first *exp*, until it passes the second *exp* by steps of the third *exp*. More precisely, a **for** statement like

```
for var = e1, e2, e3 do block end
```

is equivalent to the code:

```
do
  local _var, _limit, _step = tonumber(e1), tonumber(e2),
                              tonumber(e3)
  if not (_var and _limit and _step) then error() end
  while (_step>0 and _var<=_limit)
        or (_step<=0 and _var>=_limit) do
    local var = _var
    block
    _var = _var + _step
  end
end
```

Note the following:

— all three control expressions are evaluated only once, before the loop starts. They must all result in numbers;

— _var, _limit, and _step are invisible variables. The names are here for explanatory purposes only;

— if the third expression (the step) is absent, then a step of 1 is used;

— its possible to use **break** to exit a **for** loop;

— the loop variable `var` is local to the loop; you shall not use its value after the **for** ends or is broken. If you need the value of the loop variable `var`, then assign it to another variable before breaking or exiting the loop.

The generic **for** statement works over functions, called *iterators*. On each iteration, the iterator function is called to produce a new value, stopping when this new value is **nil**. The generic **for** loop has the following syntax:

```
stat ::= for namelist in explist1 do block end

namelist ::= Name {`,´ Name}
```


© ABNT 2011 - All rights reserved

A **for** statement like

```
for var_1, ..., var_n in explist do block end
```

is equivalent to the code:

```
do
  local _f, _s, _var = explist
  while true do
    local var_1, ... , var_n = _f(_s, _var)
    _var = var_1
    if _var == nil then break end
    block
  end
end
```

Note the following:

— explist is evaluated only once. Its results are an *iterator* function, a *state*, and an initial value for the first *iterator variable*;

— _f, _s, and _var are invisible variables. The names are here for explanatory purposes only;

— it is possible to use **break** to exit a **for** loop;

— the loop variables *var_i* are local to the loop; you shall not use their values after the **for** ends. If you need these values, then assign them to other variables before breaking or exiting the loop.

B.2.5.7 Function calls as statements

To allow possible side-effects, function calls may be executed as statements:

```
stat ::= functioncall
```

In this case, all returned values are thrown away. Function calls are explained in B.2.6.9.

B.2.5.8 Local declarations

Local variables may be declared anywhere inside a block. The declaration may include an initial assignment:

```
stat ::= local namelist [`=´ explist1]
```

If present, an initial assignment has the same semantics of a multiple assignment (see B.2.5.4). Otherwise, all variables are initialized with **nil**.

A chunk is also a block (see B.2.5.2), and so local variables may be declared in a chunk outside any explicit block. The scope of such local variables extends until the end of the chunk.

The visibility rules for local variables are explained in B.2.7.

© ABNT 2011 - All rights reserved

B.2.6 Expressions

B.2.6.1 Basic expressions

The basic expressions in Lua are the following:

```
exp ::= prefixexp
exp ::= nil  |  false  |  true
exp ::= Number
exp ::= String
exp ::= function
exp ::= tableconstructor
exp ::= `...´
exp ::= exp binop exp
exp ::= unop exp
prefixexp ::= var  |  functioncall  |  `(´ exp `)´
```

Numbers and literal strings are explained in B.2.2; variables are explained in B.2.4; function definitions are explained in B.2.6.10; function calls are explained in B.2.6.9; table constructors are explained in B.2.6.8. Vararg expressions, denoted by three dots (`...´), may only be used inside vararg functions; they are explained in B.2.6.10.

Binary operators comprise arithmetic operators (see B.2.6.2), relational operators (see B.2.6.3), and logical operators (see B.2.6.4). Unary operators comprise the unary minus (see B.2.6.2), the unary **not** (see B.2.6.4), and the unary *length operator* (see B.2.6.6).

Both function calls and vararg expressions may result in multiple values. If the expression is used as a statement (see B.2.5.7) (only possible for function calls), then its return list is adjusted to zero elements, thus discarding all returned values. If the expression is used inside another expression or in the middle of a list of expressions, then its result list is adjusted to one element, thus discarding all values except the first one. If the expression is used as the last element of a list of expressions, then no adjustment is made, unless the call is enclosed in parentheses.

Here are some examples:

```
f()        -- adjusted to 0 results
g(f(), x)  -- f() is adjusted to 1 result
g(x, f())  -- g gets x plus all values returned by f()
a,b,c = f(), x    -- f() is adjusted to 1 result (c gets nil)
a,b = ...  -- a gets the first vararg parameter, b gets
           -- the second (both a and b may get nil if
           -- there is no corresponding vararg parameter)
a,b,c = x, f()    -- f() is adjusted to 2 results
a,b,c = f()       -- f() is adjusted to 3 results
return f()-- returns all values returned by f()
return ...-- returns all received vararg parameters
return x,y,f()    -- returns x, y, and all values returned by f()
{f()}             -- creates a list with all values returned by f()
{...}             -- creates a list with all vararg parameters
{f(), nil}-- f() is adjusted to 1 result
```

An expression enclosed in parentheses always results in only one value. Thus, (f(x,y,z)) is always a single value, even if f returns several values. (The value of (f(x,y,z)) is the first value returned by f or **nil** if f does not return any values.)

© ABNT 2011 - All rights reserved

B.2.6.2 Arithmetic operators

Lua supports the usual arithmetic operators: the binary `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulo), and `^` (exponentiation); and unary `-` (negation). If the operands are numbers, or strings that may be converted to numbers (see B.2.3.2), then all operations have the usual meaning. Exponentiation works for any exponent. For instance, `x^(-0.5)` computes the inverse of the square root of `x`. Modulus is defined as

```
a % b == a - math.floor(a/b)*b
```

That is, it is the remainder of a division that rounds the quotient towards minus infinity.

B.2.6.3 Relational operators

The relational operators in Lua are

```
==   ~=   <   >   <=   >=
```

These operators always result in **false** or **true**.

Equality (`==`) first compares the type of its operands. If the types are different, then the result is **false**. Otherwise, the values of the operands are compared. Numbers and strings are compared in the usual way. Objects (tables, userdata, threads, and functions) are compared by *reference*: Two objects are considered equal only if they are the *same* object. Every time you create a new object (a table, userdata, thread, or function), this new object is different from any previously existing object.

You may change the way that Lua compares tables and userdata by using the "eq" metamethod (see B.2.9).

The conversion rules of B.2.3.2 do not apply to equality comparisons. Thus, `"0"==0` evaluates to **false**, and `t[0]` and `t["0"]` denote different entries in a table.

The operator `~=` is exactly the negation of equality (`==`).

The order operators work as follows. If both arguments are numbers, then they are compared as such. Otherwise, if both arguments are strings, then their values are compared according to the current locale. Otherwise, Lua tries to call the "lt" or the "le" metamethod (see B.2.9).

B.2.6.4 Logical operators

The logical operators in Lua are **and**, **or** and **not.** Like the control structures (see B.2.5.5), all logical operators consider both **false** and **nil** as false and anything else as true.

The negation operator **not** always returns **false** or **true**. The conjunction operator **and** returns its first argument if this value is **false** or **nil**; otherwise, **and** returns its second argument. The disjunction operator **or** returns its first argument if this value is different from **nil** and **false**; otherwise, **or** returns its second argument. Both **and** and **or** use short-cut evaluation; that is, the second operand is evaluated only if necessary. Here are some examples:

```
10 or 20            --> 10
10 or error()       --> 10
nil or "a"          --> "a"
nil and 10          --> nil
false and error()   --> false
false and nil       --> false
false or nil        --> nil
10 and 20           --> 20
```

In this Standard, `-->` indicates the result of the preceding expression.

© ABNT 2011 - All rights reserved

B.2.6.5 Concatenation

The string concatenation operator in Lua is denoted by two dots (`..´). If both operands are strings or numbers, then they are converted to strings according to the rules mentioned in B.2.3.2. Otherwise, the "concat" metamethod is called (see B.2.9).

B.2.6.6 The length operator

The length operator is denoted by the unary operator `#`. The length of a string is its number of bytes (that is, the usual meaning of string length when each character is one byte).

The length of a table `t` is defined to be any integer index `n` such that `t[n]` is not **nil** and t[n+1] is **nil**; moreover, if t[1] is **nil**, `n` may be zero. For a regular array, with non-nil values from 1 to a given `n`, its length is exactly that `n`, the index of its last value. If the array has "holes" (that is, **nil** values between other non-nil values), then `#t` may be any of the indices that directly precedes a **nil** value (that is, it may consider any such **nil** value as the end of the array).

B.2.6.7 Precedence

Operator precedence in Lua follows the table below, from lower to higher priority:

```
or
and
<     >     <=    >=    ~=    ==
..
+     -
*     /     %
not   #     - (unary)
^
```

As usual, you may use parentheses to change the precedences of an expression. The concatenation (`..´) and exponentiation (`^´) operators are right associative. All other binary operators are left associative.

B.2.6.8 Table constructors

Table constructors are expressions that create tables. Every time a constructor is evaluated, a new table is created. Constructors may be used to create empty tables, or to create a table and initialize some of its fields. The general syntax for constructors is

```
tableconstructor ::= `{´ [fieldlist] `}´
fieldlist ::= field {fieldsep field} [fieldsep]
field ::= `[´ exp `]´ `=´ exp  |  Name `=´ exp  |  exp
fieldsep ::= `,´  |  `;´
```

Each field of the form `[exp1] = exp2` adds to the new table an entry with key `exp1` and value `exp2`. A field of the form `name = exp` is equivalent to `["name"] = exp`. Finally, fields of the form `exp` are equivalent to `[i] = exp`, where `i` are consecutive numerical integers, starting with 1. Fields in the other formats do not affect this counting. For example,

```
a = { [f(1)] = g; "x", "y"; x = 1, f(x), [30] = 23; 45 }
```


© ABNT 2011 - All rights reserved

is equivalent to

```
do
  local t = {}
  t[f(1)] = g
  t[1] = "x"         -- 1st exp
  t[2] = "y"         -- 2nd exp
  t.x = 1            -- t["x"] = 1
  t[3] = f(x)        -- 3rd exp
  t[30] = 23
  t[4] = 45          -- 4th exp
  a = t
end
```

If the last field in the list has the form `exp` and the expression is a function call or a vararg expression, then all values returned by that expression enter the list consecutively (see B.2.6.9). To avoid this, enclose the function call (or the vararg expression) in parentheses (see B.2.6.1).

The field list may have an optional trailing separator, as a convenience for machine-generated code.

B.2.6.9 Function calls

A function call in Lua has the following syntax:

```
functioncall ::= prefixexp args
```

In a function call, first *prefixexp* and *args* are evaluated. If the value of *prefixexp* has type *function*, then that function is called with the given arguments. Otherwise, the *prefixexp* "call" metamethod is called, having as first parameter the value of *prefixexp*, followed by the original call arguments (see B.2.9).

The form

```
functioncall ::= prefixexp `:´ Name args
```

may be used to call "methods". A call `v:name(...)` is syntactic sugar for `v.name(v,...)`, except that `v` is evaluated only once.

Arguments have the following syntax:

```
args ::= `(´ [explist1] `)´

args ::= tableconstructor

args ::= String
```

All argument expressions are evaluated before the call. A call of the form f{...} is syntactic sugar for f({...}); that is, the argument list is a single new table. A call of the form f'...' (or f"..." or f[[...]]) is syntactic sugar for f('...'); that is, the argument list is a single literal string.

As an exception to the free-format syntax of Lua, you shall not put a line break before the `(´ in a function call. That restriction avoids some ambiguities in the language. If you write

```
a = f

g).x(a)
```

© ABNT 2011 - All rights reserved

Lua would see that as a single statement, a = f(g).x(a). So, if you want two statements, you must add a semi-colon between them. If you actually want to call f, you must remove the line break before (g).

A call of the form `return` *functioncall* is called a *tail call*. Lua implements *proper tail calls* (or *proper tail recursion*): In a tail call, the called function reuses the stack entry of the calling function. Therefore, there is no limit on the number of nested tail calls that a program may execute. However, a tail call erases any debug information about the calling function. Note that a tail call only happens with a particular syntax, where the **return** has one single function call as argument; this syntax makes the calling function return exactly the returns of the called function. So, none of the following examples are tail calls:

```
return (f(x))        -- results adjusted to 1
return 2 * f(x)
return x, f(x)       -- additional results
f(x); return         -- results discarded
return x or f(x)     -- results adjusted to 1
```

B.2.6.10 Function definitions

The syntax for function definition is

```
function ::= function funcbody

funcbody ::= `(´ [parlist1] `)´ block end
```

The following syntactic sugar simplifies function definitions:

```
stat ::= function funcname funcbody

stat ::= local function Name funcbody

funcname ::= Name {`.´ Name} [`:´ Name]
```

The statement

```
function f () ... end
```

translates to

```
f = function () ... end
```

The statement

```
function t.a.b.c.f () ... end
```

translates to

```
t.a.b.c.f = function () ... end
```

The statement

```
local function f () ... end
```

translates to

```
local f; f = function () ... end
```

not this:

```
local f = function () ... end
```


© ABNT 2011 - All rights reserved

This only makes a difference when the body of the function contains references to f.

A function definition is an executable expression, whose value has type *function*. When Lua pre-compiles a chunk, all its function bodies are pre-compiled too. Then, whenever Lua executes the function definition, the function is *instantiated* (or *closed*). This function instance (or *closure*) is the final value of the expression. Different instances of the same function may refer to different external local variables and may have different environment tables.

Parameters act as local variables that are initialized with the argument values:

```
parlist1 ::= namelist [`,´ `...´]  |  `...´
```

When a function is called, the list of arguments is adjusted to the length of the list of parameters, unless the function is a variadic or *vararg function*, which is indicated by three dots (`...´) at the end of its parameter list. A vararg function does not adjust its argument list; instead, it collects all extra arguments and supplies them to the function through a *vararg expression*, which is also written as three dots. The value of this expression is a list of all actual extra arguments, similar to a function with multiple results. If a vararg expression is used inside another expression or in the middle of a list of expressions, then its return list is adjusted to one element. If the expression is used as the last element of a list of expressions, then no adjustment is made (unless the call is enclosed in parentheses).

As an example, consider the following definitions:

```
function f(a, b) end
function g(a, b, ...) end
function r() return 1,2,3 end
```

Then, we have the following mapping from arguments to parameters and to the vararg expression:

```
CALL            PARAMETERS


f(3)             a=3, b=nil

f(3, 4)          a=3, b=4

f(3, 4, 5)       a=3, b=4

f(r(), 10)       a=1, b=10

f(r())           a=1, b=2


g(3)             a=3, b=nil, ... -->  (nothing)

g(3, 4)          a=3, b=4,   ... -->  (nothing)

g(3, 4, 5, 8)    a=3, b=4,   ... -->  5  8

g(5, r())        a=5, b=1,   ... -->  2  3
```

Results are returned using the **return** statement (see B.2.5.5). If control reaches the end of a function without encountering a **return** statement, then the function returns with no results.

© ABNT 2011 - All rights reserved

The *colon* syntax is used for defining *methods*, that is, functions that have an implicit extra parameter `self`. Thus, the statement

```
function t.a.b.c:f (...) ... end
```

is syntactic sugar for

```
t.a.b.c.f = function (self, ...) ... end
```

B.2.7 Visibility rules

Lua is a lexically scoped language. The scope of variables begins at the first statement *after* their declaration and lasts until the end of the innermost block that includes the declaration. Consider the following example:

```
x = 10                -- global variable

do                    -- new block

      local x = x         -- new `x', with value 10

      print(x)            --> 10

      x = x+1

      do                  -- another block

        local x = x+1     -- another `x'

        print(x)          --> 12

      end
      print(x)            --> 11
    end
    print(x)              --> 10  (the global one)
```

In a declaration like `local x = x`, the new `x` being declared is not in scope yet, and so the second `x` refers to the outside variable.

Because of the lexical scoping rules, local variables may be freely accessed by functions defined inside their scope. A local variable used by an inner function is called an *upvalue*, or *external local variable*, inside the inner function.

Each execution of a **local** statement defines new local variables. Consider the following example:

```
a = {}
local x = 20
for i=1,10 do
  local y = 0
  a[i] = function () y=y+1; return x+y end
end
```

The loop creates ten closures (that is, ten instances of the anonymous function). Each of these closures uses a different `y` variable, while all of them share the same `x`.

© ABNT 2011 - All rights reserved

B.2.8 Error handling

Because Lua is an embedded extension language, all Lua actions start from C code in the host program calling a function from the Lua library (see lua_pcall). Whenever an error occurs during Lua compilation or execution, control returns to C, which may take appropriate measures (such as printing an error message).

Lua code may explicitly generate an error by calling the error function. If you need to catch errors in Lua, you may use the pcall function.

B.2.9 Metatables

Every value in Lua may have a *metatable*. This *metatable* is an ordinary Lua table that defines the behavior of the original value under certain special operations. You may change several aspects of the behavior of operations over a value by setting specific fields in its metatable. For instance, when a non-numeric value is the operand of an addition, Lua checks for a function in the field `"__add"` in its metatable. If it finds one, Lua calls that function to perform the addition.

We call the keys in a metatable *events* and the values *metamethods*. In the previous example, the event is `"add"` and the metamethod is the function that performs the addition.

You may query the metatable of any value through the getmetatable function.

You may replace the metatable of tables through the setmetatable function. You shall not change the metatable of other types from Lua (except using the debug library); you must use the C API for that.

Tables and userdata have individual metatables (although multiple tables and userdata may share a same table as their metatable); values of all other types share one single metatable per type. So, there is one single metatable for all numbers, and for all strings, etc.

A metatable may control how an object behaves in arithmetic operations, order comparisons, concatenation, length operation, and indexing. A metatable may also define a function to be called when a userdata is garbage collected. For each of those operations Lua associates a specific key called an *event*. When Lua performs one of those operations over a value, it checks whether that value has a metatable with the corresponding event. If so, the value associated with that key (the *metamethod*) controls how Lua will perform the operation.

Metatables control the operations listed next. Each operation is identified by its corresponding name. The key for each operation is a string with its name prefixed by two underscores, `__´; for instance, the key for operation "add" is the string `"__add"`. The semantics of these operations is better explained by a Lua function describing how the interpreter executes that operation.

The code in Lua shown in this section is only illustrative; the real behavior is hard coded in the interpreter and it is much more efficient than this simulation. All functions used in these descriptions (rawget, tonumber, etc.) are described in B.5.2. In particular, to retrieve the metamethod of a given object, the following expression is used

```
metatable(obj)[event]
```

This shall be read as

```
rawget(getmetatable(obj) or {}, event)
```

That is, the access to a metamethod does not invoke other metamethods, and the access to objects with no metatables does not fail (it simply results in **nil**).

- **"add":** the + operation.

© ABNT 2011 - All rights reserved

The function `getbinhandler` below defines how Lua chooses a handler for a binary operation. First, Lua tries the first operand. If its type does not define a handler for the operation, then Lua tries the second operand.

```
function getbinhandler (op1, op2, event)

   return metatable(op1)[event] or metatable(op2)[event]

end
```

Using that function, the behavior of the `op1 + op2` is

```
function add_event (op1, op2)

   local o1, o2 = tonumber(op1), tonumber(op2)

   if o1 and o2 then  -- both operands are numeric?

     return o1 + o2   -- `+' here is the primitive `add'
   else  -- at least one of the operands is not numeric
     local h = getbinhandler(op1, op2, "__add")
     if h then
       -- call the handler with both operands
       return h(op1, op2)
     else  -- no handler available: default behavior
       error("...")
     end
   end
 end
```

- **"sub":** the `-` operation. Behavior similar to the "add" operation.
- **"mul":** the `*` operation. Behavior similar to the "add" operation.
- **"div":** the `/` operation. Behavior similar to the "add" operation.
- **"mod":** the `%` operation. Behavior similar to the "add" operation, with the operation o1 - floor(o1/o2)*o2 as the primitive operation.
- **"pow":** the `^` (exponentiation) operation. Behavior similar to the "add" operation, with the function `pow` (from the C math library) as the primitive operation.

© ABNT 2011 - All rights reserved

- **"unm":** the unary – operation.

```
function unm_event (op)
   local o = tonumber(op)
   if o then  -- operand is numeric?
     return -o  -- `-' here is the primitive `unm'
   else  -- the operand is not numeric.
     -- Try to get a handler from the operand
     local h = metatable(op).__unm
     if h then
       -- call the handler with the operand
       return h(op)
     else  -- no handler available: default behavior
       error("...")
     end
   end
 end
```

- **"concat":** the . . (concatenation) operation.

```
function concat_event (op1, op2)
   if (type(op1) == "string" or type(op1) == "number") and
      (type(op2) == "string" or type(op2) == "number") then
     return op1 .. op2  -- primitive string concatenation
   else
     local h = getbinhandler(op1, op2, "__concat")
     if h then
       return h(op1, op2)
     else
       error("...")
     end
   end
 end
```

- **"len":** the # operation.

```
function len_event (op)
   if type(op) == "string" then
     return strlen(op)         -- primitive string length
   elseif type(op) == "table" then
     return #op                -- primitive table length
   else
     local h = metatable(op).__len
     if h then
       -- call the handler with the operand
       return h(op)
     else  -- no handler available: default behavior
       error("...")
     end
   end
 end
```

© ABNT 2011 - All rights reserved

See B.2.6.6 for a description of the length of a table.

- **"eq":** the == operation. The function getcomphandler defines how Lua chooses a metamethod for comparison operators. A metamethod only is selected when both objects being compared have the same type and the same metamethod for the selected operation.

```
function getcomphandler (op1, op2, event)
   if type(op1) ~= type(op2) then return nil end
   local mm1 = metatable(op1)[event]
   local mm2 = metatable(op2)[event]
   if mm1 == mm2 then return mm1 else return nil end
 end
```

The "eq" event is defined as follows:

```
function eq_event (op1, op2)
  if type(op1) ~= type(op2) then  -- different types?
    return false   -- different objects
  end
  if op1 == op2 then   -- primitive equal?
    return true   -- objects are equal
  end
  -- try metamethod
  local h = getcomphandler(op1, op2, "__eq")
  if h then
    return h(op1, op2)
  else
    return false
  end
end
```

`a ~= b` is equivalent to `not (a == b)`.

- **"lt":** the < operation.

```
function lt_event (op1, op2)
   if type(op1) == "number" and type(op2) == "number" then
     return op1 < op2   -- numeric comparison
   elseif type(op1)=="string" and type(op2)=="string" then
     return op1 < op2   -- lexicographic comparison
   else
     local h = getcomphandler(op1, op2, "__lt")
     if h then
       return h(op1, op2)
     else
       error("...");
     end
   end
 end
```

a > b is equivalent to b < a.

© ABNT 2011 - All rights reserved

- **"le":** the <= operation.

```
function le_event (op1, op2)
  if type(op1) == "number" and type(op2) == "number" then
    return op1 <= op2   -- numeric comparison
  elseif type(op1)=="string" and type(op2)=="string" then
    return op1 <= op2   -- lexicographic comparison
  else
    local h = getcomphandler(op1, op2, "__le")
    if h then
      return h(op1, op2)
    else
      h = getcomphandler(op1, op2, "__lt")
      if h then
        return not h(op2, op1)
      else
        error("...");
      end
    end
  end
end
```

`a >= b` is equivalent to `b <= a`. In the absence of a "le" metamethod, Lua tries the "lt", assuming that `a <= b` is equivalent to `not (b < a)`.

- **"index":** the indexing access table[key].

```
function gettable_event (table, key)
  local h
  if type(table) == "table" then
    local v = rawget(table, key)
    if v ~= nil then return v end
    h = metatable(table).__index
    if h == nil then return nil end
  else
    h = metatable(table).__index
    if h == nil then
      error("...");
    end
  end
  if type(h) == "function" then
    return h(table, key)      -- call the handler
  else return h[key]          -- or repeat operation on it
  end
end
```

© ABNT 2011 - All rights reserved

- **"newindex":** indexing assignment table[key] = value.

```
function settable_event (table, key, value)
   local h
   if type(table) == "table" then
     local v = rawget(table, key)
     if v ~= nil then rawset(table, key, value); return end
     h = metatable(table).__newindex
     if h == nil then rawset(table, key, value); return end
   else
     h = metatable(table).__newindex
     if h == nil then
       error("...");
     end
   end
   if type(h) == "function" then
     return h(table, key,value) -- call the handler
   else h[key] = value          -- or repeat operation on it
   end
 end
```

- **"call":** called when Lua calls a value.

```
function function_event (func, ...)
   if type(func) == "function" then
     return func(...)   -- primitive call
   else
     local h = metatable(func).__call
     if h then
       return h(func, ...)
     else
       error("...")
     end
   end
 end
```

B.2.10 Environments

Besides metatables, objects of types thread, function, and userdata have another table associated with them, called their *environment*. Like metatables, environments are regular tables and multiple objects may share the same environment.

Environments associated with userdata have no meaning for Lua. It is only a feature for programmers to associate a table to a userdata.

Environments associated with threads are called global environments. They are used as the default environment for threads and non-nested functions created by that thread (through `loadfile`, `loadstring` or `load`) and may be directly accessed by C code (see B.3.4).

Environments associated with C functions may be directly accessed by C code (see B.3.4). They are used as the default environment for other C functions created by that function.

Environments associated with Lua functions are used to resolve all accesses to global variables within that function (see B.2.4). They are used as the default environment for other Lua functions created by that function.

© ABNT 2011 - All rights reserved

You may change the environment of a Lua function or the running thread by calling setfenv. You may get the environment of a Lua function or the running thread by calling getfenv. To manipulate the environment of other objects (userdata, C functions, other threads) you must use the C API.

B.2.11 Garbage collection

B.2.11.1 Basic concepts

Lua performs automatic memory management. That means that you have to worry neither about allocating memory for new objects nor about freeing it when the objects are no longer needed. Lua manages memory automatically by running a *garbage collector* from time to time to collect all *dead objects* (that is, those objects that are no longer accessible from Lua). All objects in Lua are subject to automatic management: tables, userdata, functions, threads, and strings.

Lua implements an incremental mark-and-sweep collector. It uses two numbers to control its garbage-collection cycles: the *garbage-collector pause* and the *garbage-collector step multiplier*.

The garbage-collector pause controls how long the collector waits before starting a new cycle. Larger values make the collector less aggressive. Values smaller than 1 mean the collector will not wait to start a new cycle. A value of 2 means that the collector waits more or less to double the total memory in use before starting a new cycle.

The step multiplier controls the relative speed of the collector relative to memory allocation. Larger values make the collector more aggressive but also increases the size of each incremental step. Values smaller than 1 make the collector too slow and can result in the collector never finishing a cycle. The default, 2, means that the collector runs at "twice" the speed of memory allocation.

You may change those numbers calling lua_gc in C or collectgarbage in Lua. Both get as arguments percentage points (so an argument 100 means a real value of 1). With those functions you may also get direct control of the collector (e.g., stop and restart it).

B.2.11.2 Garbage-collection metamethods

Using the C API, you may set garbage-collector metamethods for userdata (see B.2.9). These metamethods are also called finalizers. Finalizers allow you to coordinate Lua's garbage collection with external resource management (such as closing files, network or database connections, or freeing your own memory).

Garbage userdata with a field __gc in their metatables are not collected immediately by the garbage collector. Instead, Lua puts them in a list. After the collection, Lua does the equivalent of the following function for each userdata in that list:

```
function gc_event (udata)
  local h = metatable(udata).__gc
  if h then
    h(udata)
  end
end
```

At the end of each garbage-collection cycle, the finalizers for userdata are called in *reverse* order of their creation, among those collected in that cycle. That is, the first finalizer to be called is the one associated with the userdata created last in the program.

© ABNT 2011 - All rights reserved

B.2.11.3 Weak tables

A weak table is a table whose elements are weak references. A weak reference is ignored by the garbage collector. In other words, if the only references to an object are weak references, then the garbage collector will collect that object.

A weak table may have weak keys, weak values, or both. A table with weak keys allows the collection of its keys, but prevents the collection of its values. A table with both weak keys and weak values allows the collection of both keys and values. In any case, if either the key or the value is collected, the whole pair is removed from the table. The weakness of a table is controlled by the value of the __mode field of its metatable. If the __mode field is a string containing the character `k´, the keys in the table are weak. If __mode contains `v´, the values in the table are weak.

After you use a table as a metatable, you should not change the value of its field __mode. Otherwise, the weak behavior of the tables controlled by this metatable is undefined.

B.2.12 Coroutines

Lua supports coroutines, also called *collaborative multithreading*. A coroutine in Lua represents an independent thread of execution. Unlike threads in multithread systems, however, a coroutine only suspends its execution by explicitly calling a yield function.

You create a coroutine with a call to coroutine.create. Its sole argument is a function that is the main function of the coroutine. The create function only creates a new coroutine and returns a handle to it (an object of type *thread*); it does not start the coroutine execution.

When you first call coroutine.resume, passing as its first argument the thread returned by coroutine.create, the coroutine starts its execution, at the first line of its main function. Extra arguments passed to coroutine.resume are passed on to the coroutine main function. After the coroutine starts running, it runs until it terminates or *yields*.

A coroutine may terminate its execution in two ways: Normally, when its main function returns (explicitly or implicitly, after the last instruction); and abnormally, if there is an unprotected error. In the first case, coroutine.resume returns true, plus any values returned by the coroutine main function. In case of errors, coroutine.resume returns false plus an error message.

A coroutine yields by calling coroutine.yield. When a coroutine yields, the corresponding coroutine.resume returns immediately, even if the yield happens inside nested function calls (that is, not in the main function, but in a function directly or indirectly called by the main function). In the case of a yield, coroutine.resume also returns true, plus any values passed to coroutine.yield. The next time you resume the same coroutine, it continues its execution from the point where it yielded, with the call to coroutine.yield returning any extra arguments passed to coroutine.resume.

The coroutine.wrap function creates a coroutine, just like coroutine.create, but instead of returning the coroutine itself, it returns a function that, when called, resumes the coroutine. Any arguments passed to that function go as extra arguments to coroutine.resume. coroutine.wrap returns all the values returned by coroutine.resume, except the first one (the boolean error code). Unlike coroutine.resume, coroutine.wrap does not catch errors; any error is propagated to the caller.

© ABNT 2011 - All rights reserved

As an example, consider the next code:

```
function foo (a)
  print("foo", a)
  return coroutine.yield(2*a)
end

co = coroutine.create(function (a,b)
      print("co-body", a, b)
      local r = foo(a+1)
      print("co-body", r)
      local r, s = coroutine.yield(a+b, a-b)
      print("co-body", r, s)
      return b, "end"
end)

print("main", coroutine.resume(co, 1, 10))
print("main", coroutine.resume(co, "r"))
print("main", coroutine.resume(co, "x", "y"))
print("main", coroutine.resume(co, "x", "y"))
```

When you run it, it produces the following output:

```
co-body 1       10
foo     2
main    true    4
co-body r
main    true    11      -9
co-body x       y
main    true    10      end
main    false   cannot resume dead coroutine
```

B.3 Application program interface (API)

B.3.1 Basic concepts

All C API functions and related types and constants are declared in the header file `lua.h`.

Even when we use the term "function", any facility in the API may be provided as a macro instead. All such macros use each of its arguments exactly once (except for the first argument, which is always a Lua state), and so do not generate any hidden side-effects.

As in most C libraries, the Lua API functions do not check their arguments for validity or consistency. However, you may change this behavior by compiling Lua with a proper definition for the macro luai_apicheck, in file luaconf.h.

B.3.2 Stack

Lua uses a *virtual stack* to pass values to and from C. Each element in this stack represents a Lua value (nil, number, string, etc.).

Whenever Lua calls C, the called function gets a new stack, which is independent of previous stacks and of stacks of C functions that are still active. That stack initially contains any arguments to the C function and it is where the C function pushes its results to be returned to the caller (see lua_CFunction).

© ABNT 2011 - All rights reserved

For convenience, most query operations in the API do not follow a strict stack discipline. Instead, they may refer to any element in the stack by using an *index*: A positive index represents an *absolute* stack position (starting at 1); a negative index represents an *offset* relative to the top of the stack. More specifically, if the stack has *n* elements, then index 1 represents the first element (that is, the element that was pushed onto the stack first) and index *n* represents the last element; index *-1* also represents the last element (that is, the element at the top) and index *-n* represents the first element. We say that an index is *valid* if it lies between 1 and the stack top (that is, if 1 <= abs(index) <= top).

B.3.3 Stack size

When you interact with Lua API, you are responsible for ensuring consistency. In particular, *you are responsible for controlling stack overflow*. You may use the function lua_checkstack to grow the stack size.

Whenever Lua calls C, it ensures that at least LUA_MINSTACK stack positions are available. LUA_MINSTACK is defined as 20, so that usually you do not have to worry about stack space unless your code has loops pushing elements onto the stack.

Most query functions accept as indices any value inside the available stack space, that is, indices up to the maximum stack size you have set through lua_checkstack. Such indices are called *acceptable indices*. More formally, we define an *acceptable index* as follows:

```
(index < 0 && abs(index) <= top) ||
(index > 0 && index <= stackspace)
```

Note that 0 is never an acceptable index.

B.3.4 Pseudo-indices

Unless otherwise noted, any function that accepts valid indices may also be called with *pseudo-indices*, which represent some Lua values that are accessible to C code but which are not in the stack. Pseudo-indices are used to access the thread environment, the function environment, the registry, and the upvalues of a C function (see B.3.5).

The thread environment (where global variables live) is always at pseudo-index `LUA_GLOBALSINDEX`. The environment of the running C function is always at pseudo-index `LUA_ENVIRONINDEX`.

To access and change the value of global variables, you may use regular table operations over an environment table. For instance, to access the value of a global variable, do

```
lua_getfield(L, LUA_GLOBALSINDEX, varname);
```

B.3.5 C closures

When a C function is created, it is possible to associate some values with it, thus creating a *C closure*; these values are called *upvalues* and are accessible to the function whenever it is called (see lua_pushcclosure).

Whenever a C function is called, its upvalues are located at specific pseudo-indices. Those pseudo-indices are produced by the macro lua_upvalueindex. The first value associated with a function is at position lua_upvalueindex(1), and so on. Any access to lua_upvalueindex(*n*), where *n* is greater than the number of upvalues of the current function, produces an acceptable (but invalid) index.

B.3.6 Registry

Lua provides a registry, a pre-defined table that may be used by any C code to store whatever Lua value it needs to store. This table is always located at pseudo-index LUA_REGISTRYINDEX. Any C library may store data into this table, but it should take care to choose keys different from those used by other libraries, to avoid collisions. Typically, you should use as key a string containing your library name or a light userdata with the address of a C object in your code.

© ABNT 2011 - All rights reserved

The integer keys in the registry are used by the reference mechanism, implemented by the auxiliary library, and therefore should not be used for other purposes.

B.3.7 Error handling in C

Internally, Lua uses the C longjmp facility to handle errors. (You may also choose to use exceptions if you use C++; see file luaconf.h.) When Lua faces any error (such as memory allocation errors, type errors, syntax errors, and runtime errors) it *raises* an error; that is, it does a long jump. A protected environment uses setjmp to set a recover point; any error jumps to the most recent active recover point.

Almost any function in the API may raise an error, for instance due to a memory allocation error. The following functions run in protected mode (that is, they create a protected environment to run), so they never raise an error: lua_newstate, lua_close, lua_load, lua_pcall, and lua_cpcall.

Inside a C function you may raise an error by calling lua_error.

B.3.8 Functions and types

All functions and types from the C API are listed in alphabetical order. Each function has a indicator like this: [-o, +p, x].

The first field, `o`, specifies how many elements the function pops from the stack. The second field, `p`, specifies how many elements the function pushes onto the stack (any function always pushes its results after popping its arguments). A field in the form `x|y` means that the function may push (or pop) `x` or `y` elements, depending on the situation; an interrogation mark '`?`' means that we cannot know how many elements the function pops/pushes by looking only at its arguments (for example, they may depend on what is on the stack). The third field, `x`, tells whether the function may throw errors: '`-`' means the function never throws any error; '`m`' means the function may throw an error only due to not enough memory; '`e`' means the function may throw other kinds of errors; '`v`' means the function may throw an error on purpose.

---

lua_Alloc

```
typedef void * (*lua_Alloc) (void *ud, void *ptr, size_t osize, size_t nsize);
```

The type of the memory allocation function used by Lua states. The allocator function must provide a functionality similar to realloc, but not exactly the same. Its arguments are ud, an opaque pointer passed to lua_newstate; ptr, a pointer to the block being allocated/reallocated/freed; osize, the original size of the block; nsize, the new size of the block. ptr is NULL if and only if osize is zero. When nsize is zero, the allocator must return NULL; if osize is not zero, it should free the block pointed by ptr. When nsize is not zero, the allocator returns NULL if and only if it cannot fill the request. When nsize is not zero and osize is zero, the allocator should behave like malloc. When nsize and osize are not zero, the allocator behaves like realloc. Lua assumes that the allocator never fails when osize >= nsize.

Here is a simple implementation for the allocator function. It is used in the auxiliary library by lua_newstate.

```
static void *l_alloc (void *ud, void *ptr, size_t osize, size_t nsize) {
  (void)ud;       /* not used */
  (void)osize;   /* not used */
  if (nsize == 0) {
    free(ptr);  /* ANSI requires that free(NULL) has no effect */
    return NULL;
  }
  else
    /* ANSI requires that realloc(NULL, size) == malloc(size) */
    return realloc(ptr, nsize);
}
```

This code assumes that `free(NULL)` has no effect and that `realloc(NULL, size)` is equivalent to `malloc(size)`. ANSI C ensures both behaviors.

© ABNT 2011 - All rights reserved

lua_atpanic

```
lua_CFunction lua_atpanic (lua_State *L, lua_CFunction panicf);
```

Sets a new panic function and returns the old one.

If an error happens outside any protected environment, Lua calls a *panic function* and then calls exit(EXIT_FAILURE), thus exiting the host application. Your panic function can avoid this exit by never returning (for example, doing a long jump).

The panic function may access the error message at the top of the stack.

lua_call

```
void lua_call (lua_State *L, int nargs, int nresults);
```

Calls a function.

To call a function you must use the following protocol: First, the function to be called is pushed onto the stack; then, the arguments to the function are pushed in direct order; that is, the first argument is pushed first. Finally you call lua_call; nargs is the number of arguments that you pushed onto the stack. All arguments and the function value are popped from the stack when the function is called. The function results are pushed onto the stack when the function returns. The number of results is adjusted to nresults, unless nresults is LUA_MULTRET. In that case, *all* results from the function are pushed. Lua takes care that the returned values fit into the stack space. The function results are pushed onto the stack in direct order (the first result is pushed first), so that after the call the last result is on the top of the stack.

Any error inside the called function is propagated upwards (with a `longjmp`).

The following example shows how the host program can do the equivalent to this Lua code:

```
a = f("how", t.x, 14)
```

Here it is in C:

```
lua_getfield(L, LUA_GLOBALSINDEX, "f");      /* function to be called */
lua_pushstring(L, "how");                    /* 1st argument */
lua_getfield(L, LUA_GLOBALSINDEX, "t");      /* table to be indexed */
lua_getfield(L, -1, "x");                    /* push result of t.x (2nd arg) */
lua_remove(L, -2);                  /* remove `t' from the stack */
lua_pushinteger(L, 14);                      /* 3rd argument */
lua_call(L, 3, 1);                  /* call function with 3 arguments and 1 result */
lua_setfield(L, LUA_GLOBALSINDEX, "a");      /* set global variable `a' */
```

The code above is "balanced": at its end, the stack is back to its original configuration. This is considered good programming practice.

© ABNT 2011 - All rights reserved

lua_CFunction

```
typedef int (*lua_CFunction) (lua_State *L);
```

Type for C functions.

In order to communicate properly with Lua, a C function must use the following protocol, which defines the way parameters and results are passed: A C function receives its arguments from Lua in its stack in direct order (the first argument is pushed first). So, when the function starts, lua_gettop(L) returns the number of arguments received by the function. The first argument (if any) is at index 1 and its last argument is at index lua_gettop(L). To return values to Lua, a C function just pushes them onto the stack, in direct order (the first result is pushed first), and returns the number of results. Any other value in the stack below the results will be properly discarded by Lua. Like a Lua function, a C function called by Lua may also return many results.

As an example, the following function receives a variable number of numerical arguments and returns their average and sum:

```
static int foo (lua_State *L) {
  int n = lua_gettop(L);    /* number of arguments */
  lua_Number sum = 0;
  int i;
  for (i = 1; i <= n; i++) {
   if (!lua_isnumber(L, i)) {
    lua_pushstring(L,"incorrect argument to function `average'");
    lua_error(L);
   }
   sum += lua_tonumber(L, i);
  }
  lua_pushnumber(L, sum/n);        /* first result */
  lua_pushnumber(L, sum);         /* second result */
  return 2;                   /* number of results */
}
```

lua_checkstack

```
int lua_checkstack (lua_State *L, int extra);
```

Ensures that there are at least extra free stack slots in the stack. It returns false if it cannot grow the stack to that size. This function never shrinks the stack; if the stack is already larger than the new size, it is left unchanged.

lua_close

```
void lua_close (lua_State *L);
```

Destroys all objects in the given Lua state (calling the corresponding garbage-collection metamethods, if any) and frees all dynamic memory used by that state. On several platforms, you do not need to call this function, because all resources are naturally released when the host program ends. On the other hand, long-running programs, such as a daemon or a web server, might need to release states as soon as they are not needed, to avoid growing too large.

© ABNT 2011 - All rights reserved

lua_concat

```
void lua_concat (lua_State *L, int n);
```

Concatenates the n values at the top of the stack, pops them, and leaves the result at the top. If n is 1, the result is that single string (that is, the function does nothing); if n is 0, the result is the empty string. Concatenation is done following the usual semantics of Lua (see B.2.6.5).

lua_cpcall

```
int lua_cpcall (lua_State *L, lua_CFunction func, void *ud);
```

Calls the C function func in protected mode. func starts with only one element in its stack, a light userdata containing ud. In case of errors, lua_cpcall returns the same error codes as lua_pcall, plus the error object on the top of the stack; otherwise, it returns zero, and does not change the stack. All values returned by func are discarded.

lua_createtable

```
void lua_createtable (lua_State *L, int narr, int nrec);
```

Creates a new empty table and pushes it onto the stack. The new table has space pre-allocated for narr array elements and nrec non-array elements. This pre-allocation is useful when you know exactly how many elements the table will have. Otherwise you may use the function lua_newtable.

lua_dump

```
int lua_dump (lua_State *L, lua_Writer writer, void *data);
```

Dumps a function as a binary chunk. Receives a Lua function on the top of the stack and produces a binary chunk that, if loaded again, results in a function equivalent to the one dumped. As it produces parts of the chunk, lua_dump calls function writer (see lua_Writer) with the given data to write them.

The value returned is the error code returned by the last call to the writer; 0 means no errors.

This function does not pop the function from the stack.

lua_equal

```
int lua_equal (lua_State *L, int index1, int index2);
```

Returns 1 if the two values in acceptable indices index1 and index2 are equal, following the semantics of the Lua == operator (that is, may call metamethods). Otherwise returns 0. Also returns 0 if any of the indices is non valid.

lua_error

```
int lua_error (lua_State *L);
```

Generates a Lua error. The error message (which may actually be a Lua value of any type) must be on the stack top. This function does a long jump, and therefore never returns. (see luaL_error).

© ABNT 2011 - All rights reserved

lua_gc

```
int lua_gc (lua_State *L, int what, int data);
```

Controls the garbage collector.

This function performs several tasks, according to the value of the parameter `what`:

- `LUA_GCSTOP`: stops the garbage collector.
- `LUA_GCRESTART`: restarts the garbage collector.
- `LUA_GCCOLLECT`: performs a full garbage-collection cycle.
- `LUA_GCCOUNT`: returns the current amount of memory (in Kbytes) in use by Lua.
- `LUA_GCCOUNTB`: returns the remainder of dividing the current amount of bytes of memory in use by Lua by 1024.
- `LUA_GCSTEP`: performs an incremental step of garbage collection. The step "size" is controlled by `data` (larger values mean more steps) in a non-specified way. If you want to control the step size you must tune experimentally the value of `data`. The function returns 1 if that step finished a garbage-collection cycle.
- `LUA_GCSETPAUSE`: sets `data`/100 as the new value for the *pause* of the collector (see B.2.11). The function returns the previous value of the pause.
- `LUA_GCSETSTEPMUL`: sets `arg`/100 as the new value for the *step multiplier* of the collector (see B.2.11). The function returns the previous value of the step multiplier.

lua_getallocf

```
lua_Alloc lua_getallocf (lua_State *L, void **ud);
```

Returns the memory allocator function of a given state. If `ud` is not `NULL`, Lua stores in `*ud` the opaque pointer passed to lua_newstate.

lua_getfenv

```
void lua_getfenv (lua_State *L, int index);
```

Pushes on the stack the environment table of the value at the given index.

lua_getfield

```
void lua_getfield (lua_State *L, int index, const char *k);
```

Pushes onto the stack the value t[k], where t is the value at the given valid index index. As in Lua, this function may trigger a metamethod for the "index" event (see B.2.9).

lua_getglobal

```
void lua_getglobal (lua_State *L, const char *name);
```

Pushes onto the stack the value of the global name. It is defined as a macro:

```
#define lua_getglobal(L,s)  lua_getfield(L, LUA_GLOBALSINDEX, s)
```

© ABNT 2011 - All rights reserved

lua_getmetatable

```
int lua_getmetatable (lua_State *L, int index);
```

Pushes onto the stack the metatable of the value at the given acceptable index. If the index is not valid, or if the value does not have a metatable, the function returns 0 and pushes nothing on the stack.

lua_gettable

```
void lua_gettable (lua_State *L, int index);
```

Pushes onto the stack the value `t[k]`, where `t` is the value at the given valid index `index` and `k` is the value at the top of the stack.

This function pops the key from the stack (putting the resulting value in its place). As in Lua, this function may trigger a metamethod for the "index" event (see B.2.9).

lua_gettop

```
int lua_gettop (lua_State *L);
```

Returns the index of the top element in the stack. Because indices start at 1, that result is equal to the number of elements in the stack (and so 0 means an empty stack).

lua_insert

```
void lua_insert (lua_State *L, int index);
```

Moves the top element into the given valid index, shifting up the elements above that position to open space. Shall not be called with a pseudo-index, because a pseudo-index is not an actual stack position.

lua_Integer

```
typedef ptrdiff_t lua_Integer;
```

The type used by the Lua API to represent integral values.

By default it is a ptrdiff_t, which is usually the largest integral type the machine handles "comfortably".

lua_isboolean

```
int lua_isboolean (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index has type boolean, and 0 otherwise.

lua_iscfunction

```
int lua_iscfunction (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a C function, and 0 otherwise.

lua_isfunction

```
int lua_isfunction (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a function (either C or Lua), and 0 otherwise.

© ABNT 2011 - All rights reserved

lua_islightuserdata

```
int lua_islightuserdata (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a light userdata, and 0 otherwise.

lua_isnil

```
int lua_isnil (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is **nil**, and 0 otherwise.

lua_isnone

```
int lua_isnone (lua_State *L, int index);
```

Returns 1 if the given acceptable index is not valid (that is, it refers to an element outside the current stack), and 0 otherwise.

lua_isnoneornil

```
int lua_isnoneornil (lua_State *L, int index);
```

Returns 1 if the given acceptable index is not valid (that is, it refers to an element outside the current stack) or if the value at this index is **nil**, and 0 otherwise.

lua_isnumber

```
int lua_isnumber (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a number or a string convertible to a number, and 0 otherwise.

lua_isstring

```
int lua_isstring (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a string or a number (which is always convertible to a string), and 0 otherwise.

lua_istable

```
int lua_istable (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a table, and 0 otherwise.

lua_isthread

```
int lua_isthread (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a thread, and 0 otherwise.

lua_isuserdata

```
int lua_isuserdata (lua_State *L, int index);
```

Returns 1 if the value at the given acceptable index is a userdata (either full or light), and 0 otherwise.

© ABNT 2011 - All rights reserved

lua_lessthan

```
int lua_lessthan (lua_State *L, int index1, int index2);
```

Returns 1 if the value at acceptable index `index1` is smaller than the value at acceptable index `index2`, following the semantics of the Lua < operator (that is, may call metamethods). Otherwise returns 0. Also returns 0 if any of the indices is non valid.

lua_load

```
int lua_load (lua_State *L, lua_Reader reader, void *data, const char
*chunkname);
```

Loads a Lua chunk. If there are no errors, lua_load pushes the compiled chunk as a Lua function on top of the stack. Otherwise, it pushes an error message. The return values of lua_load are:

- 0 --- no errors;
- LUA_ERRSYNTAX --- syntax error during pre-compilation.
- LUA_ERRMEM --- memory allocation error.

lua_load automatically detects whether the chunk is text or binary, and loads it accordingly (see program luac).

lua_load uses a user-supplied reader function to read the chunk (see lua_Reader). The data argument is an opaque value passed to the reader function.

The chunkname argument gives a name to the chunk, which is used for error messages and in debug information (see B.3.9).

lua_newstate

```
lua_State *lua_newstate (lua_Alloc f, void *ud);
```

Creates a new, independent state. Returns NULL if cannot create the state (due to lack of memory). The argument f is the allocator function; Lua does all memory allocation for that state through that function. The second argument, ud, is an opaque pointer that Lua simply passes to the allocator in every call.

lua_newtable

```
void lua_newtable (lua_State *L);
```

Creates a new empty table and pushes it onto the stack. Equivalent to `lua_createtable(L, 0, 0)`.

lua_newthread

```
lua_State *lua_newthread (lua_State *L);
```

Creates a new thread, pushes it on the stack, and returns a pointer to a lua_State that represents this new thread. The new state returned by this function shares with the original state all global objects (such as tables), but has an independent execution stack.

There is no explicit function to close or to destroy a thread. Threads are subject to garbage collection, like any Lua object.

© ABNT 2011 - All rights reserved

lua_newuserdata

```
void *lua_newuserdata (lua_State *L, size_t size);
```

This function allocates a new block of memory with the given size, pushes on the stack a new full userdata with the block address, and returns this address.

Userdata represents C values in Lua. A *full userdata* represents a block of memory. It is an object (like a table): You must create it, it may have its own metatable, and you may detect when it is being collected. A full userdata is only equal to itself (under raw equality).

When Lua collects a full userdata with a `gc` metamethod, Lua calls the metamethod and marks the userdata as finalized. When that userdata is collected again then Lua frees its corresponding memory.

lua_next

```
int lua_next (lua_State *L, int index);
```

Pops a key from the stack, and pushes a key-value pair from the table at the given index (the "next" pair after the given key). If there are no more elements in the table, then lua_next returns 0 (and pushes nothing).

A typical traversal looks like this:

```
/* table is in the stack at index `t' */
lua_pushnil(L);  /* first key */
while (lua_next(L, t) != 0) {
  /* `key' is at index -2 and `value' at index -1 */
  printf("%s - %s\n",
    lua_typename(L, lua_type(L, -2)), lua_typename(L, lua_type(L, -1)));
  lua_pop(L, 1);  /* removes `value'; keeps `key' for next iteration */
}
```

While traversing a table, do not call lua_tolstring directly on a key, unless you know that the key is actually a string. Recall that lua_tolstring *changes* the value at the given index; this confuses the next call to lua_next.

lua_Number

```
typedef double lua_Number;
```

The type of numbers in Lua. By default, it is double, but that may be changed in luaconf.h.

Through the configuration file you may change Lua to operate with another type for numbers (for example, float or long).

lua_objlen

```
size_t lua_objlen (lua_State *L, int index);
```

Returns the "length" of the value at the given acceptable index: for strings, this is the string length; for tables, this is the result of the length operator (`#´); for userdata, this is the size of the block of memory allocated for the userdata; for other values, it is 0.

© ABNT 2011 - All rights reserved

lua_pcall

```
lua_pcall (lua_State *L, int nargs, int nresults, int errfunc);
```

Calls a function in protected mode.

Both nargs and nresults have the same meaning as in lua_call. If there are no errors during the call, lua_pcall behaves exactly like lua_call. However, if there is any error, lua_pcall catches it, pushes a single value on the stack (the error message), and returns an error code. Like lua_call, lua_pcall always removes the function and its arguments from the stack.

If errfunc is 0, then the error message returned on the stack is exactly the original error message. Otherwise, errfunc is the stack index of an *error handler function*. (In the current implementation, that index shall not be a pseudo-index.) In case of runtime errors, that function will be called with the error message and its return value will be the message returned on the stack by lua_pcall.

Typically, the error handler function is used to add more debug information to the error message, such as a stack traceback. Such information cannot be gathered after the return of lua_pcall, since by then the stack has unwound.

The lua_pcall function returns 0 in case of success or one of the following error codes (defined in `lua.h`):

- `LUA_ERRRUN`: a runtime error.
- `LUA_ERRMEM`: memory allocation error. For such errors, Lua does not call the error handler function.
- `LUA_ERRERR`: error while running the error handler function.

lua_pop

```
void lua_pop (lua_State *L, int n);
```

Pops n elements from the stack.

lua_pushboolean

```
void lua_pushboolean (lua_State *L, int b);
```

Pushes a boolean value with value b onto the stack.

lua_pushcclosure

```
void lua_pushcclosure (lua_State *L, lua_CFunction fn, int n);
```

Pushes a new C closure onto the stack.

When a C function is created, it is possible to associate some values with it, thus creating a *C closure* (see B.3.5); these values are then accessible to the function whenever it is called. To associate values with a C function, first these values should be pushed onto the stack (when there are multiple values, the first value is pushed first). Then lua_pushcclosure is called to create and push the C function onto the stack, with the argument n telling how many values should be associated with the function. lua_pushcclosure also pops these values from the stack.

© ABNT 2011 - All rights reserved

lua_pushcfunction

```
void lua_pushcfunction (lua_State *L, lua_CFunction f);
```

Pushes a C function onto the stack. This function receives a pointer to a C function and pushes on the stack a Lua value of type function that, when called, invokes the corresponding C function.

Any function to be registered in Lua must follow the correct protocol to receive its parameters and return its results (see lua_CFunction).

`lua_pushcfunction` is defined as a macro:

```
#define lua_pushcfunction(L,f)  lua_pushcclosure(L,f,0)
```

lua_pushfstring

```
const char *lua_pushfstring (lua_State *L, const char *fmt, ...);
```

Pushes onto the stack a formatted string and returns a pointer to that string. It is similar to the C function sprintf, but has some important differences:

- its not necessary to allocate the space for the result: the result is a Lua string and Lua takes care of memory allocation (and deallocation, through garbage collection);
- the conversion specifiers are quite restricted. There are no flags, widths, or precisions. The conversion specifiers may only be `%%´ (inserts a `%´ in the string), `%s´ (inserts a zero-terminated string, with no size restrictions), `%f´ (inserts a lua_Number), `%p´ (inserts a pointer as an hexadecimal numeral), `%d´ (inserts an int), and `%c´ (inserts an int as a character).

lua_pushinteger

```
void lua_pushinteger (lua_State *L, lua_Integer n);
```

Pushes a number with value n on to the stack.

lua_pushlightuserdata

```
void lua_pushlightuserdata (lua_State *L, void *p);
```

Pushes a light userdata onto the stack.

Userdata represents C values in Lua. A light userdata represents a pointer. It is a value (like a number): you do not create it, it has no metatables, it is not collected (as it was never created). A light userdata is equal to "any" light userdata with the same C address.

lua_pushlstring

```
void lua_pushlstring (lua_State *L, const char *s, size_t len);
```

Pushes the string pointed by s with size len onto the stack. Lua makes (or reuses) an internal copy of the given string, so the memory at s may be freed or reused immediately after the function returns. The string may contain embedded zeros.

lua_pushnil

```
void lua_pushnil (lua_State *L);
```

Pushes a nil value onto the stack.

© ABNT 2011 - All rights reserved

lua_pushnumber

```
void lua_pushnumber (lua_State *L, lua_Number n);
```

Pushes a number with value n onto the stack.

lua_pushstring

```
void lua_pushstring (lua_State *L, const char *s);
```

Pushes the zero-terminated string pointed by s onto the stack. Lua makes (or reuses) an internal copy of the given string, so the memory at s may be freed or reused immediately after the function returns. The string shall not contain embedded zeros; it is assumed to end at the first zero.

lua_pushthread

```
void lua_pushthread (lua_State *L);
```

Pushes the thread represented by L onto the stack. Returns 1 if this thread is the main thread of its state.

lua_pushvalue

```
void lua_pushvalue (lua_State *L, int index);
```

%%Pushes a copy of the element at the given valid index onto the stack.

lua_pushvfstring

```
const char *lua_pushvfstring (lua_State *L, const char *fmt,
                                    va_list argp);
```

Equivalent to *lua_pushfstring*, except that it receives a va_list instead of a variable number of arguments.

lua_rawequal

```
int lua_rawequal (lua_State *L, int index1, int index2);
```

Returns 1 if the two values in acceptable indices index1 and index2 are primitively equal (that is, without calling metamethods). Otherwise returns 0. Also returns 0 if any of the indices are non valid.

lua_rawget

```
void lua_rawget (lua_State *L, int index);
```

Similar to *lua_gettable*, but does a raw access (that is, without metamethods).

lua_rawgeti

```
void lua_rawgeti (lua_State *L, int index, int n);
```

Pushes onto the stack the value t[n], where t is the value at the given valid index index. The access is raw; that is, it does not invoke metamethods.

© ABNT 2011 - All rights reserved

lua_rawset

```
void lua_rawset (lua_State *L, int index);
```

Similar to lua_settable, but does a raw assignment (that is, without metamethods).

lua_rawseti

```
void lua_rawseti (lua_State *L, int index, int n);
```

Does the equivalent of t[n] = v, where t is the value at the given valid index index and v is the value at the top of the stack.

This function pops the value from the stack. The assignment is raw; that is, it does not invoke metamethods.

lua_Reader

```
typedef const char * (*lua_Reader)
                        (lua_State *L, void *data, size_t *size);
```

The reader function used by *lua_load*. Every time it needs another piece of the chunk, lua_load calls the reader, passing along its data parameter. The reader must return a pointer to a block of memory with a new piece of the chunk and set size to the block size. The block must exist until the reader function is called again. To signal the end of the chunk, the reader must return NULL. The reader function may return pieces of any size greater than zero.

lua_register

```
void lua_register (lua_State *L, const char *name, lua_CFunction f);
```

Sets the C function f as the new value of global name. It is defined as a macro:

```
#define lua_register(L,n,f)  (lua_pushcfunction(L, f), lua_setglobal(L, n))
```

lua_remove

```
void lua_remove (lua_State *L, int index);
```

Removes the element at the given valid index, shifting down the elements above that position to fill the gap. Shall not be called with a pseudo-index, because a pseudo-index is not an actual stack position.

lua_replace

```
void lua_replace (lua_State *L, int index);
```

Moves the top element into the given position (and pops it), without shifting any element (therefore replacing the value at the given position).

lua_resume

```
int lua_resume (lua_State *L, int narg);
```

Starts and resumes a coroutine in a given thread.

To start a coroutine, you first create a new thread (see *lua_newthread*); then you push on its stack the main function plus any eventual arguments; then you call *lua_resume*, with narg being the number of arguments.

© ABNT 2011 - All rights reserved

This call returns when the coroutine suspends or finishes its execution. When it returns, the stack contains all values passed to *lua_yield,* or all values returned by the body function. *lua_resume* returns *LUA_YIELD* if the coroutine yields, 0 if the coroutine finishes its execution without errors, or an error code in case of errors (see *lua_pcall*). In case of errors, the stack is not unwound, so you may use the debug API over it. The error message is on the top of the stack. To restart a coroutine, you put on its stack only the values to be passed as results from `yield`, and then call *lua_resume*.

lua_setallocf

```
void lua_setallocf (lua_State *L, lua_Alloc f, void *ud);
```

Changes the allocator function of a given state to f with user data ud.

lua_setfenv

```
int lua_setfenv (lua_State *L, int index);
```

Pops a table from the stack and sets it as the new environment for the value at the given index. If the value at the given index is neither a function nor a thread nor a userdata, *lua_setfenv* returns 0. Otherwise it returns 1.

lua_setfield

```
void lua_setfield (lua_State *L, int index, const char *k);
```

Does the equivalent to t[k] = v, where t is the value at the given valid index index and v is the value at the top of the stack,

This function pops the value from the stack. As in Lua, this function may trigger a metamethod for the "newindex" event (see B.2.9).

lua_setglobal

```
void lua_setglobal (lua_State *L, const char *name);
```

Pops a value from the stack and sets it as the new value of global name. It is defined as a macro:

```
#define lua_setglobal(L,s)   lua_setfield(L, LUA_GLOBALSINDEX, s)
```

lua_setmetatable

```
int lua_setmetatable (lua_State *L, int index);
```

Pops a table from the stack and sets it as the new metatable for the value at the given acceptable index.

lua_settable

```
void lua_settable (lua_State *L, int index);
```

Does the equivalent to t[k] = v, where t is the value at the given valid index index, v is the value at the top of the stack, and k is the value just below the top.

This function pops both the key and the value from the stack. As in Lua, this function may trigger a metamethod for the "newindex" event (see B.2.9).

© ABNT 2011 - All rights reserved

lua_settop

```
void lua_settop (lua_State *L, int index);
```

Accepts any acceptable index, or 0, and sets the stack top to that index. If the new top is larger than the old one, then the new elements are filled with **nil**. If index is 0, then all stack elements are removed.

lua_State

```
typedef struct lua_State lua_State;
```

Opaque structure that keeps the whole state of a Lua interpreter. The Lua library is fully reentrant: it has no global variables. All information about a state is kept in this structure.

A pointer to this state must be passed as the first argument to every function in the library, except to *lua_newstate*, which creates a Lua state from scratch.

lua_status

```
int lua_status (lua_State *L);
```

Returns the status of the thread L.

The status may be 0 for a normal thread, an error code if the thread finished its execution with an error, or LUA_YIELD if the thread is suspended.

lua_toboolean

```
int lua_toboolean (lua_State *L, int index);
```

Converts the Lua value at the given acceptable index to a C boolean value (0 or 1). Like all tests in Lua, *lua_toboolean* returns 1 for any Lua value different from **false** and **nil**; otherwise it returns 0. It also returns 0 when called with a non-valid index (if you want to accept only actual boolean values, use *lua_isboolean* to test the value's type).

lua_tocfunction

```
lua_CFunction lua_tocfunction (lua_State *L, int index);
```

Converts a value at the given acceptable index to a C function. That value must be a C function; otherwise, returns NULL.

lua_tointeger

```
lua_Integer lua_tointeger (lua_State *L, int idx);
```

Converts the Lua value at the given acceptable index to the signed integral type *lua_Integer*. The Lua value must be a number or a string convertible to a number (see B.2.3.2); otherwise, *lua_tointeger* returns 0.

If the number is not an integer, it is truncated in some non-specified way.

© ABNT 2011 - All rights reserved

lua_tolstring

```
const char *lua_tolstring (lua_State *L, int index, size_t *len);
```

Converts the Lua value at the given acceptable index to a string (const char*). If len is not NULL, it also sets *len with the string length. The Lua value must be a string or a number; otherwise, the function returns NULL. If the value is a number, then *lua_tolstring* also changes the actual value in the stack to a string (this change confuses *lua_next* when *lua_tolstring i*s applied to keys during a table traversal).

lua_tolstring returns a fully aligned pointer to a string inside the Lua state. This string always has a zero (`\0´) after its last character (as in C), but may contain other zeros in its body. Because Lua has garbage collection, there is no guarantee that the pointer returned by *lua_tolstring* will be valid after the corresponding value is removed from the stack.

lua_tonumber

```
lua_Number lua_tonumber (lua_State *L, int index);
```

Converts the Lua value at the given acceptable index to a number (see *lua_Number*). The Lua value must be a number or a string convertible to a number (see B.2.3.2); otherwise, *lua_tonumber* returns 0.

lua_topointer

```
const void *lua_topointer (lua_State *L, int index);
```

Converts the value at the given acceptable index to a generic C pointer (void*). The value may be a userdata, a table, a thread, or a function; otherwise, *lua_topointer* returns NULL. Lua ensures that different objects return different pointers. There is no direct way to convert the pointer back to its original value.

Typically this function is used only for debug information.

lua_tostring

```
const char *lua_tostring (lua_State *L, int index);
```

Equivalent to *lua_tolstring* with len equal to NULL.

lua_tothread

```
lua_State *lua_tothread (lua_State *L, int index);
```

Converts the value at the given acceptable index to a Lua thread (represented as lua_State*). This value must be a thread; otherwise, the function returns NULL.

lua_touserdata

```
void *lua_touserdata (lua_State *L, int index);
```

If the value at the given acceptable index is a full userdata, returns its block address. If the value is a light userdata, returns its pointer. Otherwise, returns NULL.

© ABNT 2011 - All rights reserved

lua_type

```
int lua_type (lua_State *L, int index);
```

Returns the type of the value in the given acceptable index, or LUA_TNONE for a non-valid index (that is, an index to an "empty" stack position). The types returned by *lua_type* are coded by the following constants defined in lua.h: LUA_TNIL, LUA_TNUMBER, LUA_TBOOLEAN, LUA_TSTRING, LUA_TTABLE, LUA_TFUNCTION, LUA_TUSERDATA, LUA_TTHREAD, and LUA_TLIGHTUSERDATA.

lua_typename

```
const char *lua_typename  (lua_State *L, int tp);
```

Returns the name of the type encoded by the value tp, which must be one the values returned by *lua_type*.

lua_Writer

```
typedef int (*lua_Writer) (lua_State *L, const void* p, size_t sz, void* ud);
```

The writer function used by *lua_dump*. Every time it produces another piece of chunk, *lua_dump* calls the writer, passing along the buffer to be written (p), its size (sz), and the data parameter supplied to *lua_dump*.

The writer returns an error code: 0 means no errors; any other value means an error and stops *lua_dump* from calling the writer again.

lua_xmove

```
void lua_xmove (lua_State *from, lua_State *to, int n);
```

Exchange values between different threads of the same global state.

This function pops `n` values from the stack `from`, and pushes them onto the stack `to`.

lua_yield

```
int lua_yield  (lua_State *L, int nresults);
```

Yields a coroutine.

This function should only be called as the return expression of a C function, as follows:

```
return lua_yield (L, nresults);
```

When a C function calls *lua_yield* in that way, the running coroutine suspends its execution, and the call to *lua_resume* that started this coroutine returns. The parameter nresults is the number of values from the stack that are passed as results to *lua_resume*.

B.3.9 Debug interface

Lua has no built-in debugging facilities. Instead, it offers a special interface by means of functions and hooks. This interface allows the construction of different kinds of debuggers, profilers, and other tools that need "inside information" from the interpreter.

© ABNT 2011 - All rights reserved

lua_Debug

```
typedef struct lua_Debug {
int event;
const char *name;               /* (n) */
const char *namewhat;           /* (n) */
const char *what;               /* (S) */
const char *source;             /* (S) */
int currentline;                /* (l) */
int nups;                       /* (u) number of upvalues */
int linedefined;                /* (S) */
int lastlinedefined;            /* (S) */
char short_src[LUA_IDSIZE]; /* (S) */
/* private part */
...
} lua_Debug;
```

A structure used to carry different pieces of information about an active function. `lua_getstack` fills only the private part of this structure, for later use. To fill the other fields of *lua_Debug* with useful information, call *lua_getinfo*.

The fields of *lua_Debug* have the following meaning:

- **source:** if the function was defined in a string, then `source` is that string. If the function was defined in a file, then `source` starts with a `@´ followed by the file name;
- **short_src**: a "printable" version of `source`, to be used in error messages;
- **linedefined**: the line number where the definition of the function starts;
- **lastlinedefined**: the line number where the definition of the function ends;
- **what**: the string `"Lua"` if the function is a Lua function, `"C"` if it is a C function, `"main"` if it is the main part of a chunk, and `"tail"` if it was a function that did a tail call. In the latter case, Lua has no other information about the function;
- **currentline**: the current line where the given function is executing. When no line information is available, `currentline` is set to -1;
- **name**: a reasonable name for the given function. Because functions in Lua are first-class values, they do not have a fixed name: Some functions may be the value of multiple global variables, while others may be stored only in a table field. The `lua_getinfo` function checks how the function was called to find a suitable name. If it cannot find a name, then `name` is set to `NULL`;

© ABNT 2011 - All rights reserved

- **namewhat**: explains the `name` field. The value of `namewhat` may be `"global"`, `"local"`, `"method"`, `"field"`, `"upvalue"`, or `""` (the empty string), according to how the function was called (Lua uses the empty string when no other option seems to apply);
- **nups**: the number of upvalues of the function.

---

lua_gethook

```
lua_Hook lua_gethook (lua_State *L);
```

Returns the current hook function.

---

lua_gethookcount

```
int lua_gethookcount (lua_State *L);
```

Returns the current hook count.

---

lua_gethookmask

```
int lua_gethookmask (lua_State *L);
```

Returns the current hook mask.

---

lua_getinfo

```
int lua_getinfo (lua_State *L, const char *what, lua_Debug *ar);
```

Returns information about a specific function or function invocation.

To get information about a function invocation,the parameter ar shall be a valid activation record that wasfilled by a revious call to lua_getstack orgiven as argument to a hook (see lua_Hook).

To get information about a function you push it onto the stackand start the what string with the character '>' (in that case, lua_getinfo pops the function in the top of the stack). For instance, to know in which line a function f was defined,you can write the following code:

```
lua_Debug ar;

lua_getfield(L, LUA_GLOBALSINDEX, "f"); /* get global 'f' */

lua_getinfo(L, ">S", &ar);

printf("%d\n", ar.linedefined);
```

Each character in the string whatselects some fields of the structure ar to be filled ora value to be pushed on the stack:

- **'n'**: fills in the field name and namewhat;
- **'S'**: fills in the fields source, short_src,linedefined, lastlinedefined, and what;
- **'l'**: fills in the field currentline;
- **'u**': fills in the field nups;
- **'f'**: pushes onto the stack the function that isrunning at the given level;

© ABNT 2011 - All rights reserved

- **‘L’**: pushes onto the stack a table whose indices are thenumbers of the lines that are valid on the function (a valid line is a line with some associated code,that is, a line where you can put a break point.Non-valid lines include empty lines and comments).

This function returns 0 on error (for instance, an invalid option in what).

---

lua_getlocal

```
const char *lua_getlocal (lua_State *L, const lua_Debug *ar, int n);
```

Gets information about a local variable of a given activation record. The parameter ar must be a valid activation record that was filled by a previous call to *lua_getstack* or given as argument to a hook (see *lua_Hook*). The index n selects which local variable to inspect (1 is the first parameter or active local variable, and so on, until the last active local variable). *lua_getlocal* pushes the variable's value onto the stack and returns its name.

Variable names starting with `(´ (open parentheses) represent internal variables (loop control variables, temporaries, and C function locals).

Returns `NULL` (and pushes nothing) when the index is greater than the number of active local variables.

---

lua_getstack

```
int lua_getstack (lua_State *L, int level, lua_Debug *ar);
```

Get information about the interpreter runtime stack.

This function fills parts of a *`lua_Debug`* structure with an identification of the *activation record* of the function executing at a given level. Level 0 is the current running function, whereas level *n+1* is the function that has called level *n*. When there are no errors, *`lua_getstack`* returns 1; when called with a level greater than the stack depth, it returns 0.

---

lua_getupvalue

```
const char *lua_getupvalue (lua_State *L, int funcindex, int n);
```

Gets information about a closure's upvalue. (For Lua functions, upvalues are the external local variables that the function uses, and that consequently are included in its closure.) *lua_getupvalue* gets the index n of an upvalue, pushes the upvalue's value onto the stack, and returns its name. funcindex points to the closure in the stack (upvalues have no particular order, as they are active through the whole function. So, they are numbered in an arbitrary order).

Returns `NULL` (and pushes nothing) when the index is greater than the number of upvalues. For C functions, this function uses the empty string `""` as a name for all upvalues.

---

lua_Hook

```
typedef void (*lua_Hook) (lua_State *L, lua_Debug *ar);
```

Type for debugging hook functions.

Whenever a hook is called, its `ar` argument has its field `event` set to the specific event that triggered the hook. Lua identifies these events with the following constants: `LUA_HOOKCALL`, `LUA_HOOKRET`, `LUA_HOOKTAILRET`, `LUA_HOOKLINE`, and `LUA_HOOKCOUNT`. Moreover, for line events, the field `currentline` is also set. To get the value of any other field in `ar`, the hook must call *`lua_getinfo`*. For return events, `event` may be `LUA_HOOKRET`, the normal value, or `LUA_HOOKTAILRET`. In the latter case, Lua is simulating a return from a function that did a tail call; in this case, it is useless to call *`lua_getinfo`*.

© ABNT 2011 - All rights reserved

While Lua is running a hook, it disables other calls to hooks. Therefore, if a hook calls back Lua to execute a function or a chunk, that execution occurs without any calls to hooks.

lua_sethook

```
int lua_sethook (lua_State *L, lua_Hook func, int mask,  int count);
```

Sets the debugging hook function.

`func` is the hook function. `mask` specifies on which events the hook will be called: it is formed by a bitwise or of the constants `LUA_MASKCALL`, `LUA_MASKRET`, `LUA_MASKLINE`, and `LUA_MASKCOUNT`. The `count` argument is only meaningful when the mask includes `LUA_MASKCOUNT`. For each event, the hook is called as explained below:

- **the call hook** is called when the interpreter calls a function. The hook is called just after Lua enters the new function, before the function gets its arguments;
- **the return hook** is called when the interpreter returns from a function. The hook is called just before Lua leaves the function. You have no access to the values to be returned by the function;
- **the line hook** is called when the interpreter is about to start the execution of a new line of code, or when it jumps back in the code (even to the same line). (This event only happens while Lua is executing a Lua function);
- **the count hook** is called after the interpreter executes every `count` instructions. (This event only happens while Lua is executing a Lua function).

A hook is disabled by setting `mask` to zero.

lua_setlocal

```
const char *lua_setlocal (lua_State *L, const lua_Debug *ar, int n);
```

Sets the value of a local variable of a given activation record. Parameters ar and n are as in *lua_getlocal* (see *lua_getlocal*). *lua_setlocal* assigns the value at the top of the stack to the variable and returns its name. It also pops the value from the stack.

Returns `NULL` (and pops nothing) when the index is greater than the number of active local variables.

lua_setupvalue

```
const char *lua_setupvalue (lua_State *L, int funcindex, int n);
```

Sets the value of a closure's upvalue. Parameters funcindex and n are as in *lua_getupvalue* (see *lua_getupvalue*). It assigns the value at the top of the stack to the upvalue and returns its name. It also pops the value from the stack.

Returns `NULL` (and pops nothing) when the index is greater than the number of upvalues.

© ABNT 2011 - All rights reserved

B.4 Auxiliary library

B.4.1 Basic concepts

The auxiliary library provides several convenient functions to interface C with Lua. While the basic API provides the primitive functions for all interactions between C and Lua, the auxiliary library provides higher-level functions for some common tasks.

All functions from the auxiliary library are defined in header file `lauxlib.h` and have a prefix `luaL_`.

All functions in the auxiliary library are built on top of the basic API, and so they provide nothing that cannot be done with that API.

Several functions in the auxiliary library are used to check C function arguments. Their names are always `luaL_check*` or `luaL_opt*`. All of these functions raise an error if the check is not satisfied. Because the error message is formatted for arguments (for example, `"bad argument #1"`), you should not use these functions for other stack values.

B.4.2 Functions and types

All functions and types from the auxiliary library in alphabetical order.

---

luaL_addchar

```
void luaL_addchar (luaL_Buffer B, char c);
```

Adds the character c to the buffer B (see *luaL_Buffer*).

---

luaL_addlstring

```
void luaL_addlstring (luaL_Buffer *B, const char *s, size_t l);
```

Adds the string pointed by s with length l to the buffer B (see *luaL_Buffer*). The string may contain embedded zeros.

---

luaL_addsize

```
void luaL_addsize (luaL_Buffer B, size_t n);
```

Adds a string of length n previously copied to the buffer area (see *luaL_prepbuffer*) to the buffer B (see *luaL_Buffer*).

---

luaL_addstring

```
void luaL_addstring (luaL_Buffer *B, const char *s);
```

Adds the zero-terminated string pointed by `s` to the buffer `B` (see *`luaL_Buffer`*). The string shall not contain embedded zeros.

© ABNT 2011 - All rights reserved

luaL_addvalue

```
void luaL_addvalue (luaL_Buffer *B);
```

Adds the value at the top of the stack to the buffer B (see *luaL_Buffer*). Pops the value.

This is the only function on string buffers that shall be called with an extra element on the stack, which is the value to be added to the buffer.

luaL_argcheck

```
void luaL_argcheck (lua_State *L, int cond, int numarg, const char *extramsg);
```

Checks whether cond is true. If not, raises an error with the following message, where func is retrieved from the call stack:

```
bad argument #<numarg> to <func> <extramsg>)
```

luaL_argerror

```
int luaL_argerror (lua_State *L, int numarg, const char *extramsg);
```

Raises an error with the following message, where func is retrieved from the call stack:

```
bad argument #<numarg> to <func> (<extramsg>)
```

This function never returns, but it is an idiom to use it as `return luaL_argerror(args)` in C functions.

luaL_Buffer

```
typedef struct luaL_Buffer luaL_Buffer;
```

Type for a *string buffer*.

A string buffer allows C code to build Lua strings piecemeal. Its pattern of use is as follows:

- first you declare a variable `b` of type *`luaL_Buffer`*;
- then you initialize it with a call `luaL_buffinit(L, &b);`
- then you add string pieces to the buffer calling any of the `luaL_add*` functions;
- you finish by calling `luaL_pushresult(&b)`. That call leaves the final string on the top of the stack.

During its normal operation, a string buffer uses a variable number of stack slots. So, while using a buffer, you shall not assume that you know where the top of the stack is. You may use the stack between successive calls to buffer operations as long as that use is balanced; that is, when you call a buffer operation, the stack is at the same level it was immediately after the previous buffer operation (the only exception to this rule is *`luaL_addvalue`*). After calling *`luaL_pushresult`* the stack is back to its level when the buffer was initialized, plus the final string on its top.

luaL_buffinit

```
void luaL_buffinit (lua_State *L, luaL_Buffer *B);
```

Initializes a buffer B. This function does not allocate any space; the buffer must be declared as a variable (see *luaL_Buffer*).

© ABNT 2011 - All rights reserved

luaL_callmeta

```
int luaL_callmeta (lua_State *L, int obj, const char *e);
```

Calls a metamethod.

If the object at index `obj` has a metatable and that metatable has a field `e`, this function calls that field and passes the object as its only argument. In that case this function returns 1 and pushes on the stack the value returned by the call. If there is no metatable or no metamethod, this function returns 0 (without pushing any value on the stack).

luaL_checkany

```
void luaL_checkany (lua_State *L, int narg);
```

Checks whether the function has an argument of any type (including **nil**) at position `narg`.

luaL_checkint

```
int luaL_checkint (lua_State *L, int narg);
```

Checks whether the function argument narg is a number and returns that number cast to an int.

luaL_checkinteger

```
lua_Integer luaL_checkinteger (lua_State *L, int narg);
```

Checks whether the function argument `narg` is a number and returns that number cast to a *`lua_Integer`*.

luaL_checklong

```
long luaL_checklong (lua_State *L, int narg);
```

Checks whether the function argument narg is a number and returns that number cast to a long.

luaL_checklstring

```
const char *luaL_checklstring (lua_State *L, int narg, size_t *l);
```

Checks whether the function argument narg is a string and returns that string; if l is not NULL fills *l with the string's length.

luaL_checknumber

```
lua_Number luaL_checknumber (lua_State *L, int narg);
```

Checks whether the function argument narg is a number and returns that number.

luaL_checkoption

```
int luaL_checkoption (lua_State *L, int narg, const char *def, const char *const
lst[]);
```

Checks whether the function argument narg is a string and searches for that string into the array lst (which must be NULL-terminated). If def is not NULL, uses def as a default value when the function has no argument narg or if that argument is **nil**.

© ABNT 2011 - All rights reserved

Returns the index in the array where the string was found. Raises an error if the argument is not a string or if the string cannot be found.

This is a useful function for mapping strings to C enums. The usual convention in Lua libraries is to use strings instead of numbers to select options.

luaL_checkstack

```
void luaL_checkstack (lua_State *L, int sz, const char *msg);
```

Grows the stack size to top + sz elements, raising an error if the stack cannot grow to that size. msg is an additional text to go into the error message.

luaL_checkstring

```
const char *luaL_checkstring (lua_State *L, int narg);
```

Checks whether the function argument narg is a string and returns that string.

luaL_checktype

```
void luaL_checktype (lua_State *L, int narg, int t);
```

Checks whether the function argument narg has type t.

luaL_checkudata

```
void *luaL_checkudata (lua_State *L, int narg, const char *tname);
```

Checks whether the function argument narg is a userdata of the type tname (see *luaL_newmetatable*).

luaL_dofile

```
luaL_dofileint luaL_dofile (lua_State *L, const char *filename);
```

Loads and runs the given file.It is defined as the following macro:

```
(luaL_loadfile(L, filename) || lua_pcall(L, 0, LUA_MULTRET, 0))
```

It returns 0 if there are no errorsor 1 in case of errors.

luaL_dostring

```
luaL_dostringint luaL_dostring (lua_State *L, const char *str);
```

Loads and runs the given string.It is defined as the following macro:

```
(luaL_loadstring(L, str) || lua_pcall(L, 0, LUA_MULTRET, 0))
```

It returns 0 if there are no errorsor 1 in case of errors.

© ABNT 2011 - All rights reserved

luaL_error

```
int luaL_error (lua_State *L, const char *fmt, ...);
```

Raises an error. The error message format is given by fmt plus any extra arguments, following the same rules of *lua_pushfstring*. It also adds at the beginning of the message the file name and the line number where the error occurred, if that information is available.

This function never returns, but it is an idiom to use it as `return luaL_error(args)` in C functions.

luaL_getmetafield

```
int luaL_getmetafield (lua_State *L, int obj, const char *e);
```

Pushes on the stack the field e from the metatable of the object at index obj. If the object does not have a metatable, or if the metatable does not have that field, returns 0 and pushes nothing.

luaL_getmetatable

```
void luaL_getmetatable (lua_State *L, const char *tname);
```

Pushes on the stack the metatable associated to name tname in the registry (see *luaL_newmetatable*).

luaL_gsub

```
const char *luaL_gsub (lua_State *L, const char *s, const char *p, const char *r);
```

Creates a copy of string s by replacing any occurrence of the string p with the string r. Pushes the resulting string on the stack and returns it.

luaL_loadbuffer

```
int luaL_loadbuffer (lua_State *L, const char *buff, size_t sz, const char
*name);
```

Loads a buffer as a Lua chunk. This function uses *lua_load* to load the chunk in the buffer pointed by buff with size sz.

This function returns the same results as *lua_load*. `name` is the chunk name, used for debug information and error messages.

luaL_loadfile

```
int luaL_loadfile (lua_State *L, const char *filename);
```

Loads a file as a Lua chunk. This function uses *lua_load* to load the chunk in the file named filename. If filename is NULL, then it loads from the standard input. The first line in the file is ignored if it starts with a #.

This function returns the same results as *lua_load*, but it has an extra error code `LUA_ERRFILE` if it cannot open/read the file.

As lua_load, this function only loads the chunk;it does not run it.

© ABNT 2011 - All rights reserved

luaL_loadstring

```
int luaL_loadstring (lua_State *L, const char *s);
```

Loads a string as a Lua chunk. This function uses *lua_load* to load the chunk in the zero-terminated string s.

This function returns the same results as *lua_load*.

Also as lua_load, this function only loads the chunk;it does not run it.

luaL_newmetatable

```
int luaL_newmetatable (lua_State *L, const char *tname);
```

If the registry already has the key tname, returns 0. Otherwise, creates a new table to be used as a metatable for userdata, adds it to the registry with key tname, and returns 1.

In both cases pushes on the stack the final value associated with `tname` in the registry.

luaL_newstate

```
lua_State *luaL_newstate (void);
```

Creates a new Lua state, calling *lua_newstate* with an allocation function based on the standard C realloc function and setting a panic function (see *lua_atpanic*) that prints an error message to the standard error output in case of fatal errors.

Returns the new state, or `NULL` if there is a memory allocation error.

luaL_openlibs

```
void luaL_openlibs (lua_State *L);
```

Opens all standard Lua libraries into the given state.

luaL_optint

```
int luaL_optint (lua_State *L, int narg, int d);
```

If the function argument narg is a number, returns that number cast to an int. If that argument is absent or is **nil**, returns d. Otherwise, raises an error.

luaL_optinteger

```
lua_Integer luaL_optinteger (lua_State *L, int narg, lua_Integer d);
```

If the function argument narg is a number, returns that number cast to a *lua_Integer*. If that argument is absent or is **nil**, returns d. Otherwise, raises an error.

luaL_optlong

```
long luaL_optlong (lua_State *L, int narg, long d);
```

If the function argument narg is a number, returns that number cast to a long. If that argument is absent or is **nil**, returns d. Otherwise, raises an error.

© ABNT 2011 - All rights reserved

luaL_optlstring

```
const char *luaL_optlstring (lua_State *L, int narg, const char *d, size_t *l);
```

If the function argument narg is a string, returns that string. If that argument is absent or is **nil**, returns d. Otherwise, raises an error.

If `l` is not `NULL`, fills the position `*l` with the results's length.

luaL_optnumber

```
lua_Number luaL_optnumber (lua_State *L, int narg, lua_Number d);
```

If the function argument narg is a number, returns that number. If that argument is absent or is **nil**, returns d. Otherwise, raises an error.

luaL_optstring

```
const char *luaL_optstring (lua_State *L, int narg, const char *d);
```

If the function argument narg is a string, returns that string. If that argument is absent or is **nil**, returns d. Otherwise, raises an error.

luaL_prepbuffer

```
char *luaL_prepbuffer (luaL_Buffer *B);
```

Returns an address to a space of size LUAL_BUFFERSIZE where you may copy a string to be added to buffer B (see *luaL_Buffer*). After copying the string into that space you must call *luaL_addsize* with the size of the string to actually add it to the buffer.

luaL_pushresult

```
void luaL_pushresult (luaL_Buffer *B);
```

Finishes the use of buffer B leaving the final string on the top of the stack.

luaL_ref

```
int luaL_ref (lua_State *L, int t);
```

Creates and returns a reference, in the table at index t, for the object at the top of the stack (and pops the object).

A reference is a unique integer key. As long as you do not manually add integer keys into table `t`, *`luaL_ref`* ensures the uniqueness of the key it returns. You may retrieve an object referred by reference `r` by calling `lua_rawgeti(L, t, r)`. Function *`luaL_unref`* frees a reference and its associated object.

If the object at the top of the stack is **nil**, *`luaL_ref`* returns the constant `LUA_REFNIL`. The constant `LUA_NOREF` is guaranteed to be different from any reference returned by *`luaL_ref`*.

© ABNT 2011 - All rights reserved

luaL_Reg

```
typedef struct luaL_Reg {
  const char *name;
  lua_CFunction func;
} luaL_Reg;
```

Type for arrays of functions to be registered by *luaL_register*. name is the function name and func is a pointer to the function. Any array of *luaL_Reg* must end with an sentinel entry in which both name and func are NULL.

luaL_register

```
void luaL_register (lua_State *L, const char *libname, const luaL_Reg *l);
```

Opens a library.

When called with `libname` equal to `NULL`, simply registers all functions in the list `l` (see *`luaL_Reg`*) into the table on the top of the stack.

When called with a non-null `libname`, creates a new table `t`, sets it as the value of the global variable `libname`, sets it as the value of `package.loaded[libname]`, and registers on it all functions in the list `l`. If there is a table in `package.loaded[libname]` or in variable `libname`, reuses that table instead of creating a new one.

In any case the function leaves the table on the top of the stack.

luaL_typename

```
const char *luaL_typename (lua_State *L, int idx);
```

Returns the name of the type of the value at index idx.

luaL_typerror

```
int luaL_typerror (lua_State *L, int narg, const char *tname);
```

Generates an error with a message like

```
<location>: bad argument <narg> to <function> (<tname> expected, got <realt>)
```

where `<location>` is produced by *`luaL_where`*, `<function>` is the name of the current function, and `<realt>` is the type name of the actual argument.

luaL_unref

```
void luaL_unref (lua_State *L, int t, int ref);
```

Releases reference ref from the table at index t (see luaL_ref). The entry is removed from the table, so that the referred object may be collected. The reference ref is also freed to be used again.

If `ref` is `LUA_NOREF` or `LUA_REFNIL`, `luaL_unref` does nothing.

luaL_where

```
void luaL_where (lua_State *L, int lvl);
```

Pushes on the stack a string identifying the current position of the control at level lvl in the call stack. Typically this string has the format <chunkname>:<currentline>:

© ABNT 2011 - All rights reserved

Level 0 is the running function, level 1 is the function that called the running function, etc.

This function is used to build a prefix for error messages.

B.5 Standard libraries

B.5.1 Overview

The standard Lua libraries provide useful functions that are implemented directly through the C API. Some of these functions provide essential services to the language (for example, *type* and *getmetatable*); others provide access to "outside" services (for example, I/O); and others could be implemented in Lua itself, but are quite useful or have critical performance requirements that deserve an implementation in C (for example, *table*.*sort*).

All libraries are implemented through the official C API and are provided as separate C modules. Currently, Lua has the following standard libraries:

- basic library;
- package library;
- string manipulation;
- table manipulation;
- mathematical functions (sin, log, etc.);
- input and output;
- operating system facilities;
- debug facilities.

Except for the basic and package libraries, each library provides all its functions as fields of a global table or as methods of its objects.

To have access to these libraries, the C host program shall call `luaL_openlibs`, which open all standard libraries. Alternatively, it may open them individually by calling `luaopen_base` (for the basic library), `luaopen_package` (for the package library), `luaopen_string` (for the string library), `luaopen_table` (for the table library), `luaopen_math` (for the mathematical library), `luaopen_io` (for the I/O and the Operating System libraries), and `luaopen_debug` (for the debug library). These functions are declared in `lualib.h` and should not be called directly: you shall call them like any other Lua C function, for example, by using `lua_call`.

B.5.2 Basic functions

The basic library provides some core functions to Lua. If you do not include this library in your application, you shall check carefully whether you need to provide implementations for some of its facilities.

---

assert (v [, message])

Issues an error when the value of its argument `v` is false (that is, **nil** or **false**); otherwise, returns all its arguments. `message` is an error message; when absent, it defaults to "assertion failed!"

© ABNT 2011 - All rights reserved

collectgarbage (opt [, arg])

This function is a generic interface to the garbage collector. It performs different functions according to its first argument, opt:

- **"stop"**: stops the garbage collector.
- **"restart"**: restarts the garbage collector.
- **"collect"**: performs a full garbage-collection cycle.
- **"count"**: returns the total memory in use by Lua (in Kbytes).
- **"step"**: performs a garbage-collection step. The step "size" is controlled by `arg` (larger values mean more steps) in a non-specified way. If you want to control the step size you shall tune experimentally the value of `arg`. Returns **true** if that step finished a collection cycle.
- **"steppause"**: sets `arg`/100 as the new value for the *pause* of the collector (see B.2.11).
- **"setstepmul"**: sets `arg`/100 as the new value for the *step multiplier* of the collector (see B.2.11).

dofile (filename)

Opens the named file and executes its contents as a Lua chunk. When called without arguments, `dofile` executes the contents of the standard input (`stdin`). Returns all values returned by the chunk. In case of errors, `dofile` propagates the error to its caller (that is, `dofile` does not run in protected mode).

error (message [, level])

Terminates the last protected function called and returns `message` as the error message. Function `error` never returns.

Usually, `error` adds some information about the error position at the beginning of the message. The `level` argument specifies how to get the error position. With level 1 (the default), the error position is where the `error` function was called. Level 2 points the error to where the function that called `error` was called; and so on. Passing a level 0 avoids the addition of error position information to the message.

_G

A global variable (not a function) that holds the global environment (that is, `_G._G = _G`). Lua itself does not use this variable; changing its value does not affect any environment, nor vice-versa (use *`setfenv`* to change environments).

getfenv (f)

Returns the current environment in use by the function. `f` may be a Lua function or a number that specifies the function at that stack level: Level 1 is the function calling `getfenv`. If the given function is not a Lua function, or if `f` is 0, `getfenv` returns the global environment. The default for `f` is 1.

getmetatable (object)

If `object` does not have a metatable, returns **nil**. Otherwise, if the object's metatable has a `"__metatable"` field, returns the associated value. Otherwise, returns the metatable of the given object.

© ABNT 2011 - All rights reserved

ipairs (t)

Returns three values: an iterator function, the table `t`, and 0, so that the construction

```
for i,v in ipairs(t) do ... end
```

will iterate over the pairs (`1,t[1]`), (`2,t[2]`), ..., up to the first integer key with a nil value in the table.

load (func [, chunkname])

Loads a chunk using function `func` to get its pieces. Each call to `func` must return a string that concatenates with previous results. A return of **nil** (or no value) signals the end of the chunk.

If there are no errors, returns the compiled chunk as a function; otherwise, returns **nil** plus the error message. The environment of the returned function is the global environment.

`chunkname` is used as the chunk name for error messages and debug information. When absent,it defaults to "=(load)".

loadfile ([filename])

Similar to *`load`*, but gets the chunk from file `filename` or from the standard input, if no file name is given.

loadstring (string [, chunkname])

Similar to *`load`*, but gets the chunk from the given string.

To load and run a given string, use the idiom

```
assert(loadstring(s))()
```

When absent,chunkname defaults to the given string.

next (table [, index])

Allows a program to traverse all fields of a table. Its first argument is a table and its second argument is an index in this table. `next` returns the next index of the table and its associated value. When called with **nil** as its second argument, `next` returns an initial index and its associated value. When called with the last index, or with **nil** in an empty table, `next` returns **nil**. If the second argument is absent, then it is interpreted as **nil**. In particular, you may use `next(t)` to check whether a table is empty.

The order in which the indices are enumerated is not specified, even for numeric indices (to traverse a table in numeric order,use a numerical **for** or the ipairs function).

The behavior of `next` is undefined if, during the traversal, you assign any value to a non-existent field in the table. You may however modify existing fields. In particular, you may clear existing fields.

pairs (t)

Returns three values: the *`next`* function, the table `t`, and **nil**, so that the construction

```
for k,v in pairs(t) do ... end
```

will iterate over all key--value pairs of table `t`.

See *`next`* for the caveats of modifying the table during its traversal.

© ABNT 2011 - All rights reserved

pcall (f, arg1, arg2, ...)

Calls function `f` with the given arguments in protected mode. That means that any error inside `f` is not propagated; instead, `pcall` catches the error and returns a status code. Its first result is the status code (a boolean), which is true if the call succeeds without errors. In such case, `pcall` also returns all results from the call, after this first result. In case of any error, `pcall` returns **false** plus the error message.

print (e1, e2, ...)

Receives any number of arguments, and prints their values in `stdout`, using the *`tostring`* function to convert them to strings. `print` is not intended for formatted output, but only as a quick way to show a value, typically for debugging. For formatted output, use *`string.format`*.

rawequal (v1, v2)

Checks whether `v1` is equal to `v2`, without invoking any metamethod. Returns a boolean.

rawget (table, index)

Gets the real value of `table[index]`, without invoking any metamethod. `table` must be a table and `index` any value different from **nil**.

rawset (table, index, value)

Sets the real value of `table[index]` to `value`, without invoking any metamethod. `table` must be a table, `index` any value different from **nil**, and `value` any Lua value.

This function returns `table`.

select (index, ...)

If `index` is a number, returns all arguments after argument number `index`. Otherwise, `index` must be the string `"#"`, and `select` returns the total number of extra arguments it received.

setfenv (f, table)

Sets the environment to be used by the given function. `f` may be a Lua function or a number that specifies the function at that stack level: level 1 is the function calling `setfenv`. `setfenv` returns the given function.

As a special case, when `f` is 0 `setfenv` changes the environment of the running thread. In this case, `setfenv` returns no values.

setmetatable (table, metatable)

Sets the metatable for the given table (you shall not change the metatable of other types from Lua, only from C). If `metatable` is **nil**, removes the metatable of the given table. If the original metatable has a `"__metatable"` field, raises an error.

This function returns `table`.

© ABNT 2011 - All rights reserved

tonumber (e [, base])

Tries to convert its argument to a number. If the argument is already a number or a string convertible to a number, then `tonumber` returns that number; otherwise, it returns **nil**.

An optional argument specifies the base to interpret the numeral. The base may be any integer between 2 and 36, inclusive. In bases above 10, the letter `A´ (in either upper or lower case) represents 10, `B´ represents 11, and so forth, with `Z´ representing 35. In base 10 (the default), the number may have a decimal part, as well as an optional exponent part (see B.2.2). In other bases, only unsigned integers are accepted.

tostring (e)

Receives an argument of any type and converts it to a string in a reasonable format. For complete control of how numbers are converted, use *`string.format`*.

If the metatable of `e` has a "`__tostring`" field, then `tostring` calls the corresponding value with `e` as argument, and uses the result of the call as its result.

type (v)

Returns the type of its only argument, coded as a string. The possible results of this function are "`nil`" (a string, not the value **nil**), "`number`", "`string`", "`boolean`, "`table`", "`function`", "`thread`", and "`userdata`".

unpack (list [, i [, j]])

Returns the elements from the given table. This function is equivalent to

```
return list[i], list[i+1], ..., list[j]
```

except that the above code may be written only for a fixed number of elements. By default, `i` is 1 and `j` is the length of the list, as defined by the length operator (see B.2.6.6).

_VERSION

A global variable (not a function) that holds a string containing the current interpreter version. The current contents of this variable is "`Lua 5.1`".

xpcall (f, err)

This function is similar to `pcall`, except that you may set a new error handler.

`xpcall` calls function `f` in protected mode, using `err` as the error handler. Any error inside `f` is not propagated; instead, `xpcall` catches the error, calls the `err` function with the original error object, and returns a status code. Its first result is the status code (a boolean), which is true if the call succeeds without errors. In this case, `xpcall` also returns all results from the call, after this first result. In case of any error, `xpcall` returns **false** plus the result from `err`.

© ABNT 2011 - All rights reserved

B.5.3 Coroutine manipulation

The operations related to coroutines comprise a sub-library of the basic library and come inside the table `coroutine`. See B.2.12 for a general description of coroutines.

---

coroutine.create (f)

Creates a new coroutine, with body `f`. `f` must be a Lua function. Returns this new coroutine, an object with type "thread".

---

coroutine.resume (co [, val1, ..., valn])

Starts or continues the execution of coroutine `co`. The first time you resume a coroutine, it starts running its body. The values `val1`, ..., `valn` are passed as the arguments to the body function. If the coroutine has yielded, `resume` restarts it; the values `val1`, ..., `valn` are passed as the results from the yield.

If the coroutine runs without any errors, `resume` returns **true** plus any values passed to `yield` (if the coroutine yields) or any values returned by the body function (if the coroutine terminates). If there is any error, `resume` returns **false** plus the error message.

---

coroutine.running ()

Returns the running coroutine, or **nil** when called by the main thread.

---

coroutine.status (co)

Returns the status of coroutine `co`, as a string: `"running"`, if the coroutine is running (that is, it called `status`); `"suspended"`, if the coroutine is suspended in a call to `yield`, or if it has not started running yet; `"normal"` if the coroutine is active but not running (that is, it has resumed another coroutine); and `"dead"` if the coroutine has finished its body function, or if it has stopped with an error.

---

coroutine.wrap (f)

Creates a new coroutine, with body `f`. `f` must be a Lua function. Returns a function that resumes the coroutine each time it is called. Any arguments passed to the function behave as the extra arguments to `resume`. Returns the same values returned by `resume`, except the first boolean. In case of error, propagates the error.

---

coroutine.yield ([val1, ..., valn])

Suspends the execution of the calling coroutine. The coroutine may be running neither a C function, nor a metamethod, nor an iterator. Any arguments to `yield` are passed as extra results to `resume`.

© ABNT 2011 - All rights reserved

B.5.4 Modules

The package library provides basic facilities for loading and building modules in Lua. It exports two of its functions directly in the global environment: *require* and *module*. Everything else is exported in a table package.

---

module (name [, ...])

Creates a module. If there is a table in package.loaded[name], that table is the module. Otherwise, if there is a global table t with the given name, that table is the module. Otherwise creates a new table t and sets it as the value of the global name and the value of package.loaded[name]. This function also initializes t._NAME with the given name, t._M with the module (t itself), and t._PACKAGE with the package name (the full module name minus last component; see below). Finally, module sets t as the new environment of the current function and the new value of package.loaded[name], so that *require* returns t.

If name is a compound name (that is, one with components separated by dots), module creates (or reuses, if they already exist) tables for each component. For instance, if name is a.b.c, then module stores the module table in field c of field b of global a.

This function may receive options after the module name, where each option is a function to be applied over the module.

---

require (modname)

Loads the given module. The function starts by looking into the table package.loaded to determine whether modname is already loaded. If it is, then require returns the value stored at package.loaded[modname]. Otherwise, it tries to find a loader for that module.

To find a loader, require is guided by the package.loaders array. By changing this array, we can change how require looks for a module. The following explanation is based on the default configurationfor package.loaders.

First require queries package.preload[modname]. If it has a value,this value (which should be a function) is the loader.Otherwise require searches for a Lua loader using thepath stored in package.path.If that also fails, it searches for a C loader using thepath stored in package.cpath. If that also fails,it tries an all-in-one loader (see package.loaders).

Once a loader is found, require calls the loader with a single argument, modname. If the loader returns any value, require assigns it to package.loaded[modname]. If the loader returns no value and has not assigned any value to package.loaded[modname], then require assigns **true** to that entry. In any case, require returns the final value of package.loaded[modname].

If there is any error loading or running the module, or if it cannot find any loader for that module, then require signals an error.

---

package.cpath

The path used by *require* to search for a C loader.

Lua initializes the C path package.cpath in the same way it initializes the Lua path *package.path*, using the environment variable LUA_CPATH or another default path defined in luaconf.h.

---

package.loaded

A table used by *require* to control which modules are already loaded. When you require a module modname and package.loaded[modname] is not false, *require* simply returns the value stored there.

© ABNT 2011 - All rights reserved

package.loaders

A table used by require to control how to load modules.

Each entry in this table is a searcher function. When looking for a module, require calls each of these searchers in ascending order, with the module name (the argument given to require) as itssole parameter. The function may return another function (the module loader) or a string explaining why it did not find that module (or nil if it has nothing to say). Lua initializes this table with four functions.

The first searcher simply looks for a loader in thepackage.preload table.

The second searcher looks for a loader as a Lua library, using the path stored at package.path. A path is a sequence of templates separated by semicolons. For each template,the searcher will change each interrogationmark in the template by filename, which is the module name with each dot replaced by a "directory separator" (such as "/" in Unix); then it will try to open the resulting file name. For instance, if the Lua path is the string

```
"./?.lua;./?.lc;/usr/local/?/init.lua"
```

the search for a Lua file for module foowill try to open the files./foo.lua, ./foo.lc, and/usr/local/foo/init.lua, in that order.

The third searcher looks for a loader as a C library,using the path given by the variable package.cpath. For instance, if the C path is the string

```
"./?.so;./?.dll;/usr/local/?/init.so"
```

the searcher for module foowill try to open the files ./foo.so, ./foo.dll,and /usr/local/foo/init.so, in that order. Once it finds a C library,this searcher first uses a dynamic link facility to link theapplication with the library. Then it tries to find a C function inside the library tobe used as the loader. The name of this C function is the string "luaopen_"concatenated with a copy of the module name where each dotis replaced by an underscore. Moreover, if the module name has a hyphen,its prefix up to (and including) the first hyphen is removed. For instance, if the module name is a.v1-b.c, the function name will be luaopen_b_c.

The fourth searcher tries an all-in-one loader. It searches the C path for a library forthe root name of the given module. For instance, when requiring a.b.c,it will search for a C library for a. If found, it looks into it for an open function forthe submodule; in the example, that would be luaopen_a_b_c. With this facility, a package can pack several C submodulesinto one single library,with each submodule keeping its original open function.

package.loadlib (libname, funcname)

Dynamically links the host program with the C library `libname`. Inside this library, looks for a function `funcname` and returns this function as a C function (so, `funcname` shall follow the protocol (see *`lua_CFunction`*).

This is a low-level function. It completely bypasses the package and module system. Unlike *`require`*, it does not perform any path searching and does not automatically adds extensions. `libname` shall be the complete file name of the C library, including if necessary a path and extension. `funcname` shall be the exact name exported by the C library (which may depend on the C compiler and linker used).

This function is not supported by ANSI C. As such, it is only available on some platforms (Windows, Linux, Mac OS X, Solaris, BSD, plus other Unix systems that support the `dlfcn` standard).

© ABNT 2011 - All rights reserved

package.path

The path used by require to search for a Lua loader.

At start-up, Lua initializes this variable withthe value of the environment variable LUA_PATH orwith a default path defined in luaconf.h,if the environment variable is not defined. Any ";;" in the value of the environment variableis replaced by the default path.

package.preload

A table to store loaders for specific modules (see *require*).

package.seeall (module)

Sets a metatable for `module` with its `__index` field referring to the global environment, so that this module inherits values from the global environment. To be used as an option to function *module*.

B.5.5 String manipulation

This library provides generic functions for string manipulation, such as finding and extracting substrings, and pattern matching. When indexing a string in Lua, the first character is at position 1 (not at 0, as in C). Indices are allowed to be negative and are interpreted as indexing backwards, from the end of the string. Thus, the last character is at position - 1, and so on.

The string library provides all its functions inside the table `string`. It also sets a metatable for strings where the `__index` field points to the metatable itself. Therefore, you may use the string functions in object-oriented style. For instance, `string.byte(s, i)` may be written as `s:byte(i)`.

string.byte (s [, i [, j]])

Returns the internal numerical codes of the characters `s[i]`, `s[i+1]`, ..., `s[j]`. The default value for `i` is 1; the default value for `j` is `i`.

Numerical codes are not necessarily portable across platforms.

string.char (i1, i2, ...)

Receives 0 or more integers. Returns a string with length equal to the number of arguments, in which each character has the internal numerical code equal to its corresponding argument.

Numerical codes are not necessarily portable across platforms.

string.dump (function)

Returns a string containing a binary representation of the given function, so that a later *loadstring* on that string returns a copy of the function. `function` must be a Lua function without upvalues.

string.find (s, pattern [, init [, plain]])

Looks for the first match of `pattern` in the string `s`. If it finds a match, then `find` returns the indices of `s` where this occurrence starts and ends; otherwise, it returns **nil**. A third, optional numerical argument `init` specifies where to start the search; its default value is 1 and may be negative. A value of **true** as a fourth, optional argument `plain` turns off the pattern matching facilities, so the function does a plain "find substring" operation, with no characters in `pattern` being considered "magic". If `plain` is given, then `init` shall be given as well.

If the pattern has captures, then in a successful match the captured values are also returned, after the two indices.

© ABNT 2011 - All rights reserved

string.format (formatstring, e1, e2, ...)

Returns a formatted version of its variable number of arguments following the description given in its first argument (which must be a string). The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the options/modifiers `*`, `l`, `L`, `n`, `p`, and `h` are not supported and that there is an extra option, `q`. The `q` option formats a string in a form suitable to be safely read back by the Lua interpreter: The string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written. For instance, the call

```
string.format('%q', 'a string with "quotes" and \n new line')
```

will produce the string:

```
"a string with \"quotes\" and \
 new line"
```

The options `c`, `d`, `E`, `e`, `f`, `g`, `G`, `i`, `o`, `u`, `X`, and `x` all expect a number as argument, whereas `q` and `s` expect a string.

This function does not accept string valuescontaining embedded zeros,except as arguments to the q option.

string.gmatch (s, pattern)

Returns an iterator function that, each time it is called, returns the next captures from `pattern` over string `s`.

If `pattern` specifies no captures, then the whole match is produced in each call. As an example, the following loop

```
s = "hello world from Lua"
for w in string.gmatch(s, "%a+") do
  print(w)
end
```

will iterate over all the words from string `s`, printing one per line. The next example collects all pairs `key=value` from the given string into a table:

```
t = {}
s = "from=world, to=Lua"
for k, v in string.gmatch(s, "(%w+)=(%w+)") do
   t[k] = v
 end
```

For this function, a '^' at the start of a pattern does notwork as an anchor, as this would prevent the iteration.

string.gsub (s, pattern, repl [, n])

Returns a copy of `s` in which all occurrences of the `pattern` have been replaced by a replacement string specified by `repl`, which may be a string, a table, or a function. `gsub` also returns, as its second value, the total number of substitutions made.

If `repl` is a string, then its value is used for replacement. The character `%` works as an escape character: Any sequence in `repl` of the form `%`*n*, with *n* between 1 and 9, stands for the value of the *n*-th captured substring (see below). The sequence `%0` stands for the whole match. The sequence `%%` stands for a single `%`.

© ABNT 2011 - All rights reserved

If `repl` is a table, then the table is queried for every match, using the first capture as the key; if the pattern specifies no captures, then the whole match is used as the key.

If `repl` is a function, then this function is called every time a match occurs, with all captured substrings passed as arguments, in order; if the pattern specifies no captures, then the whole match is passed as a sole argument.

If the value returned by the table query or by the function call is a string or a number, then it is used as the replacement string; otherwise, if it is **false** or **nil**, then there is no replacement (that is, the original match is kept in the string).

Here are some examples:

```
x = string.gsub("hello world", "(%w+)", "%1 %1")

--> x="hello hello world world"


x = string.gsub("hello world", "%w+", "%0 %0", 1)

--> x="hello hello world"


x = string.gsub("hello world from Lua", "(%w+)%s*(%w+)","%2 %1")

--> x="world hello Lua from"


x = string.gsub("home = $HOME, user = $USER", "%$(%w+)",os.getenv)

--> x="home = /home/roberto, user = roberto"


x = string.gsub("4+5 = $return 4+5$", "%$(.-)%$", function (s)

      return loadstring(s)()

    end)

--> x="4+5 = 9"


local t = {name="lua", version="5.1"}

x = string.gsub("$name%-$version.tar.gz", "%$(%w+)", t)

--> x="lua-5.1.tar.gz"
```

string.len (s)

Receives a string and returns its length. The empty string `""` has length 0. Embedded zeros are counted, so `"a\000bc\000"` has length 5.

© ABNT 2011 - All rights reserved

string.lower (s)

Receives a string and returns a copy of that string with all uppercase letters changed to lowercase. All other characters are left unchanged. The definition of what an uppercase letter is depends on the current locale.

string.match (s, pattern [, init])

Looks for the first match of `pattern` in the string `s`. If it finds one, then `match` returns the captures from the pattern; otherwise it returns **nil**. If `pattern` specifies no captures, then the whole match is returned. A third, optional numerical argument `init` specifies where to start the search; its default value is 1 and may be negative.

string.rep (s, n)

Returns a string that is the concatenation of `n` copies of the string `s`.

string.reverse (s)

Returns a string that is the string `s` reversed.

string.sub (s, i [, j])

Returns the substring of `s` that starts at `i` and continues until `j`; `i` and `j` may be negative. If `j` is absent, then it is assumed to be equal to *-1* (which is the same as the string length). In particular, the call `string.sub(s,1,j)` returns a prefix of `s` with length `j`, and `string.sub(s, -i)` returns a suffix of `s` with length `i`.

string.upper (s)

Receives a string and returns a copy of that string with all lowercase letters changed to uppercase. All other characters are left unchanged. The definition of what a lowercase letter is depends on the current locale.

B.5.6 Patterns

A character class is used to represent a set of characters. The following combinations are allowed in describing a character class:

- ***x***: (where *x* is not one of the *magic characters* `^$()%.[]*+-?`) --- represents the character *x* itself.
- **`.`** : (a dot) represents all characters.
- **`%a`**: represents all letters.
- **`%c`**: represents all control characters.
- **`%d`**: represents all digits.
- **`%l`**: represents all lowercase letters.
- **`%p`**: represents all punctuation characters.
- **`%s`**: represents all space characters.
- **`%u`**: represents all uppercase letters.
- **`%w`**: represents all alphanumeric characters.

© ABNT 2011 - All rights reserved

- **`%x`**: represents all hexadecimal digits.
- **`%z`**: represents the character with representation 0.
- **`%`*x***: (where *x* is any non-alphanumeric character) --- represents the character *x*. This is the standard way to escape the magic characters. Any punctuation character (even the non magic) may be preceded by a `%´ when used to represent itself in a pattern.
- **`[`*set*`]`**: represents the class which is the union of all characters in *set*. A range of characters may be specified by separating the end characters of the range with a `-´. All classes `%`*x* described above may also be used as components in *set*. All other characters in *set* represent themselves. For example, `[%w_]` (or `[_%w]`) represents all alphanumeric characters plus the underscore, `[0-7]` represents the octal digits, and `[0-7%l%-]` represents the octal digits plus the lowercase letters plus the `-´ character.

  The interaction between ranges and classes is not defined. Therefore, patterns like `[%a-z]` or `[a-%%]` have no meaning.
- **`[^`*set*`]`**: represents the complement of *set*, where *set* is interpreted as above.

For all classes represented by single letters (`%a`, `%c`, etc.), the corresponding uppercase letter represents the complement of the class. For instance, `%S` represents all non-space characters.

The definitions of letter, space, and other character groups depend on the current locale. In particular, the class `[a-z]` cannot be equivalent to `%l`.

A pattern item may be

- a single character class, which matches any single character in the class;
- a single character class followed by `*´, which matches 0 or more repetitions of characters in the class. These repetition items will always match the longest possible sequence;
- a single character class followed by `+´, which matches 1 or more repetitions of characters in the class. These repetition items will always match the longest possible sequence;
- a single character class followed by `-´, which also matches 0 or more repetitions of characters in the class. Unlike `*´, these repetition items will always match the *shortest* possible sequence;
- a single character class followed by `?´, which matches 0 or 1 occurrence of a character in the class;
- `%`*n*, for *n* between 1 and 9; such item matches a substring equal to the *n*-th captured string (see below);
- `%b`*xy*, where *x* and *y* are two distinct characters; such item matches strings that start with *x*, end with *y*, and where the *x* and *y* are balanced. This means that, if one reads the string from left to right, counting *+1* for an *x* and *-1* for a *y*, the ending *y* is the first *y* where the count reaches 0. For instance, the item `%b()` matches expressions with balanced parentheses.

A pattern is a sequence of pattern items. A `^´ at the beginning of a pattern anchors the match at the beginning of the subject string. A `$´ at the end of a pattern anchors the match at the end of the subject string. At other positions, `^´ and `$´ have no special meaning and represent themselves.

A pattern may contain sub-patterns enclosed in parentheses; they describe *captures*. When a match succeeds, the substrings of the subject string that match captures are stored (captured) for future use. Captures are numbered according to their left parentheses. For instance, in the pattern `"(a*(.)%w(%s*))"`, the part of the string matching `"a*(.)%w(%s*)"` is stored as the first capture (and therefore has number 1); the character matching `"."` is captured with number 2, and the part matching `"%s*"` has number 3.

© ABNT 2011 - All rights reserved

As a special case, the empty capture `()` captures the current string position (a number). For instance, if we apply the pattern `"()aa()"` on the string `"flaaap"`, there will be two captures: 3 and 5.

A pattern shall not contain embedded zeros. Use `%z` instead.

B.5.7 Table manipulation

This library provides generic functions for table manipulation. It provides all its functions inside the table `table`.

Most functions in the table library assume that the table represents an array or a list. For those functions, when we talk about the "length" of a table we mean the result of the length operator.

table.concat (table [, sep [, i [, j]]])

Given an array where all elements are strings or numbers, returns table[i]..sep..table[i+1] ··· sep..table[j]. The default value for sep is the empty string, the default for i is 1, and the default for j is the length of the table. If i is greater than j, returns the empty string.

table.insert (table, [pos,] value)

Inserts element `value` at position `pos` in `table`, shifting up other elements to open space, if necessary. The default value for `pos` is `n+1`, where `n` is the length of the table (see B.2.6.6), so that a call `table.insert(t,x)` inserts `x` at the end of table `t`.

table.maxn (table)

Returns the largest positive numerical index of the given table, or zero if the table has no positive numerical indices. (To do its job this function does a linear traversal of the whole table).

table.remove (table [, pos])

Removes from `table` the element at position `pos`, shifting down other elements to close the space, if necessary. Returns the value of the removed element. The default value for `pos` is `n`, where `n` is the length of the table, so that a call `table.remove(t)` removes the last element of table `t`.

table.sort (table [, comp])

Sorts table elements in a given order, *in-place*, from `table[1]` to `table[n]`, where `n` is the length of the table. If `comp` is given, then it must be a function that receives two table elements, and returns true when the first is less than the second (so that `not comp(a[i+1],a[i])` will be true after the sort). If `comp` is not given, then the standard Lua operator `<` is used instead.

The sort algorithm is not stable; that is, elements considered equal by the given order may have their relative positions changed by the sort.

© ABNT 2011 - All rights reserved

B.5.8 Mathematical functions

This library is an interface to the standard C math library. It provides all its functions inside the table math.

---

math.abs (x)

Returns the absolute value of x.

---

math.acos (x)

Returns the arc cosine of x (in radians).

---

math.asin (x)

Returns the arc sine of x (in radians).

---

math.atan (x)

Returns the arc tangent of x (in radians).

---

math.atan2 (y, x)

Returns the arc tangent of y/x (in radians), but uses the signs of both parameters to find the quadrant of the result (it also handles correctly the case of x being zero).

---

math.ceil (x)

Returns the smallest integer larger than or equal to x.

---

math.cos (x)

Returns the cosine of x (assumed to be in radians).

---

math.cosh (x)

Returns the hyperbolic cosine of x.

© ABNT 2011 - All rights reserved

math.deg (x)

Returns the angle x (given in radians) in degrees.

math.exp (x)

Returns the value e^x.

math.floor (x)

Returns the largest integer smaller than or equal to x.

math.fmod (x, y)

Returns the remainder of the division of x by y.

math.frexp (x)

Returns m and e such that x = $m2^e$, e is an integer and the absolute value of m is in the range [0.5, 1) (or zero when x is zero).

math.huge (x)

The value HUGE_VAL, a value larger than or equal to any other numerical value.

math.ldexp (m, e)

Returns $m2^e$ (e shall be an integer).

math.log (x)

Returns the natural logarithm of x.

math.log10 (x)

Returns the base - 10 logarithm of x.

math.max (x, ···)

Returns the maximum value among its arguments.

© ABNT 2011 - All rights reserved

math.min (x, ···)

Returns the minimum value among its arguments.

math.modf (x)

Returns two numbers, the integral part of x and the fractional part of x.

math.pi

The value of *pi*.

math.pow (x, y)

Returns x^y (you can also use the expression x^y to compute this value).

math.rad (x)

Returns the angle x (given in degrees) in radians.

math.random ([m [, n]])

This function is an interface to the simple pseudo-random generator function rand provided by ANSI C (no guarantees can be given for its statistical properties).

When called without arguments, returns a uniform pseudo-random real number in the range [0,1]. When called with an integer number m, math.random returns a uniform pseudo-random integer in the range [1, m]. When called with two integer numbers m and n, math.random returns a uniform pseudo-random integer in the range [m, n].

math.randomseed (x)

Sets x as the "seed" for the pseudo-random generator: equal seeds produce equal sequences of numbers.

math.sin (x)

Returns the sine of x (assumed to be in radians).

`math.sinh (x)`

Returns the hyperbolic sine of x.

© ABNT 2011 - All rights reserved

math.sqrt (x)

Returns the square root of x (you can also use the expression x^0.5 to compute this value).

math.tan (x)

Returns the tangent of x (assumed to be in radians).

math.tanh (x)

Returns the hyperbolic tangent of x.

B.5.9 Input and output facilities

The I/O library provides two different styles for file manipulation. The first one uses implicit file descriptors; that is, there are operations to set a default input file and a default output file, and all input/output operations are over those default files. The second style uses explicit file descriptors.

When using implicit file descriptors, all operations are supplied by table `io`. When using explicit file descriptors, the operation *`io.open`* returns a file descriptor and then all operations are supplied as methods of the file descriptor.

The table `io` also provides three predefined file descriptors with their usual meanings from C: `io.stdin`, `io.stdout`, and *`io.stderr`*.

Unless otherwise stated, all I/O functions return **nil** on failure (plus an error message as a second result) and some value different from **nil** on success.

io.close ([file])

Equivalent to `file:close()`. Without a `file`, closes the default output file.

io.flush ()

Equivalent to `file:flush` over the default output file.

io.input ([file])

When called with a file name, it opens the named file (in text mode), and sets its handle as the default input file. When called with a file handle, it simply sets that file handle as the default input file. When called without parameters, it returns the current default input file.

In case of errors this function raises the error, instead of returning an error code.

io.lines ([filename])

Opens the given file name in read mode and returns an iterator function that, each time it is called, returns a new line from the file. Therefore, the construction

```
for line in io.lines(filename) do ... end
```

will iterate over all lines of the file. When the iterator function detects the end of file, it returns **nil** (to finish the loop) and automatically closes the file.

The call `io.lines()` (without a file name) is equivalent to `io.input():lines()`; that is, it iterates over the lines of the default input file. In that case it does not close the file when the loop ends.

© ABNT 2011 - All rights reserved

io.open (filename [, mode])

This function opens a file, in the mode specified in the string `mode`. It returns a new file handle, or, in case of errors, **nil** plus an error message.

The `mode` string may be any of the following:

- **"r"**: read mode (the default);
- **"w"**: write mode;
- **"a"**: append mode;
- **"r+"**: update mode, all previous data is preserved;
- **"w+"**: update mode, all previous data is erased;
- **"a+"**: append update mode, previous data is preserved, writing is only allowed at the end of file.

The `mode` string may also have a `b´ at the end, which is needed in some systems to open the file in binary mode. This string is exactly what is used in the standard C function `fopen`.

io.output ([file])

Similar to *`io.input`*, but operates over the default output file.

io.popen ([prog [, mode]])

Starts program `prog` in a separated process and returns a file handle that you may use to read data from that program (if `mode` is `"r"`, the default) or to write data to that program (if `mode` is `"w"`).

This function is system dependent and is not available on all platforms.

io.read (format1, ...)

Equivalent to `io.input():read`.

io.tmpfile ()

Returns a handle for a temporary file. This file is opened in update mode and it is automatically removed when the program ends.

io.type (obj)

Checks whether `obj` is a valid file handle. Returns the string `"file"` if `obj` is an open file handle, `"closed file"` if `obj` is a closed file handle, and **nil** if `obj` is not a file handle.

io.write (value1, ...)

Equivalent to `io.output():write`.

file:close ()

Closes `file`. Note that files are automatically closed when their handles are garbage collected, but that takes an unpredictable amount of time to happen.

© ABNT 2011 - All rights reserved

file:flush ()

Saves any written data to `file`.

file:lines ()

Returns an iterator function that, each time it is called, returns a new line from the file. Therefore, the construction

```
for line in file:lines() do ... end
```

will iterate over all lines of the file. (Unlike `io.lines`, this function does not close the file when the loop ends.)

file:read (format1, ...)

Reads the file `file`, according to the given formats, which specify what to read. For each format, the function returns a string (or a number) with the characters read, or **nil** if it cannot read data with the specified format. When called without formats, it uses a default format that reads the entire next line (see below).

The available formats are

- **"*n"**: reads a number; this is the only format that returns a number instead of a string.
- **"*a"**: reads the whole file, starting at the current position. On end of file, it returns the empty string.
- **"*l"**: reads the next line (skipping the end of line), returning **nil** on end of file. This is the default format.
- ***number***: reads a string with up to that number of characters, returning **nil** on end of file. If number is zero, it reads nothing and returns an empty string, or **nil** on end of file.

file:seek ([whence] [, offset])

Sets and gets the file position, measured from the beginning of the file, to the position given by `offset` plus a base specified by the string `whence`, as follows:

- **"set"**: base is position 0 (beginning of the file);
- **"cur"**: base is current position;
- **"end"**: base is end of file;

In case of success, function `seek` returns the final file position, measured in bytes from the beginning of the file. If this function fails, it returns **nil**, plus a string describing the error.

The default value for `whence` is `"cur"`, and for `offset` is 0. Therefore, the call `file:seek()` returns the current file position, without changing it; the call `file:seek("set")` sets the position to the beginning of the file (and returns 0); and the call `file:seek("end")` sets the position to the end of the file, and returns its size.

file:setvbuf (mode [, size])

Sets the buffering mode for an output file. There are three available modes:

- **"no"**: no buffering; the result of any output operation appears immediately.
- **"full"**: full buffering; output operation is performed only when the buffer is full (or when you explicitly `flush` the file (see *io.flush*)).

© ABNT 2011 - All rights reserved

- **"line"**: line buffering; output is buffered until a newline is output or there is any input from some special files (such as a terminal device).

For the last two cases, `sizes` specifies the size of the buffer, in bytes. The default is an appropriate size.

---

file:write (value1, ...)

Writes the value of each of its arguments to the `file`. The arguments must be strings or numbers. To write other values, use *`tostring`* or *`string.format`* before `write`.

B.5.10 Operating system facilities

This library is implemented through table `os`.

---

os.clock ()

Returns an approximation of the amount in seconds of CPU time used by the program.

---

os.date ([format [, time]])

Returns a string or a table containing date and time, formatted according to the given string `format`.

If the `time` argument is present, this is the time to be formatted (see the `os.time` function for a description of this value). Otherwise, `date` formats the current time.

If `format` starts with `!´, then the date is formatted in Coordinated Universal Time. After that optional character, if `format` is `*t`, then `date` returns a table with the following fields: `year` (four digits), `month` (1--12), `day` (1--31), `hour` (0--23), `min` (0--59), `sec` (0--61), `wday` (weekday, Sunday is 1), `yday` (day of the year), and `isdst` (daylight saving flag, a boolean).

If `format` is not `*t`, then `date` returns the date as a string, formatted according to the same rules as the C function `strftime`.

When called without arguments, `date` returns a reasonable date and time representation that depends on the host system and on the current locale (that is, `os.date()` is equivalent to `os.date("%c")`).

---

os.difftime (t2, t1)

Returns the number of seconds from time `t1` to time `t2`. In POSIX, Windows, and some other systems, this value is exactly `t2-t1`.

---

os.execute ([command])

This function is equivalent to the C function `system`. It passes `command` to be executed by an operating system shell. It returns a status code, which is system-dependent. If `command` is absent, then it returns nonzero if a shell is available and zero otherwise.

---

os.exit ([code])

Calls the C function `exit`, with an optional `code`, to terminate the host program. The default value for `code` is the success code.

---

os.getenv (varname)

Returns the value of the process environment variable `varname`, or **nil** if the variable is not defined.

© ABNT 2011 - All rights reserved

os.remove (filename)

Deletes the file or directory with the given name. Directories must be empty to be removed. If this function fails, it returns **nil**, plus a string describing the error.

os.rename (oldname, newname)

Renames file or directory named `oldname` to `newname`. If this function fails, it returns **nil**, plus a string describing the error.

os.setlocale (locale [, category])

Sets the current locale of the program. `locale` is a string specifying a locale; `category` is an optional string describing which category to change: `"all"`, `"collate"`, `"ctype"`, `"monetary"`, `"numeric"`, or `"time"`; the default category is `"all"`. The function returns the name of the new locale, or **nil** if the request cannot be honored.

If locale is the empty string, the current locale is set to an implementation-defined native locale. If locale is the string "C", the current locale is set to the standard C locale.

When called with **nil** as the first argument, this function only returns the name of the current locale for the given category.

os.time ([table])

Returns the current time when called without arguments, or a time representing the date and time specified by the given table. This table must have fields `year`, `month`, and `day`, and may have fields `hour`, `min`, `sec`, and `isdst` (for a description of these fields, see the *`os.date`* function).

The returned value is a number, whose meaning depends on your system. In POSIX, Windows, and some other systems, this number counts the number of seconds since some given start time (the "epoch"). In other systems, the meaning is not specified, and the number returned by `time` may be used only as an argument to `date` and `difftime`.

os.tmpname ()

Returns a string with a file name that may be used for a temporary file. The file must be explicitly opened before its use and explicitly removed when no longer needed.

B.5.11 Debug library

This library provides the functionality of the debug interface to Lua programs. You should exert care when using this library. The functions provided here should be used exclusively for debugging and similar tasks, such as profiling. Shall be avoid to use them as a usual programming tool: they can be very slow. Moreover, several of its functions violate some assumptions about Lua code (for example, that variables local to a function cannot be accessed from outside or that userdata metatables cannot be changed by Lua code) and therefore can compromise otherwise secure code.

All functions in this library are provided inside the `debug` table. All functions that operate over a thread have an optional first argument which is the thread to operate over. The default is always the current thread.

debug.debug ()

Enters an interactive mode with the user, running each string that the user enters. Using simple commands and other debug facilities, the user may inspect global and local variables, change their values, evaluate expressions, and so on. A line containing only the word `cont` finishes this function, so that the caller continues its execution.

The commands for `debug.debug` are not lexically nested within any function, and so have no direct access to local variables.

© ABNT 2011 - All rights reserved

debug.getfenv (o)

Returns the environment of object `o`.

debug.gethook ()

Returns the current hook settings, as three values: the current hook function, the current hook mask, and the current hook count (as set by the *`debug.sethook`* function).

debug.getinfo (function [, what])

Returns a table with information about a function. You may give the function directly, or you may give a number as the value of `function`, which means the function running at level `function` of the call stack: Level 0 is the current function (`getinfo` itself); level 1 is the function that called `getinfo`; and so on. If `function` is a number larger than the number of active functions, then `getinfo` returns **nil**.

The returned table contains all the fields returned by *`lua_getinfo`*, with the string `what` describing which fields to fill in. The default for `what` is to get all information available. If present, the option `f´ adds a field named `func` with the function itself.

For instance, the expression `debug.getinfo(1,"n").name` returns a name of the current function, if a reasonable name can be found, and `debug.getinfo(print)` returns a table with all available information about the *`print`* function.

debug.getlocal (level, local)

This function returns the name and the value of the local variable with index `local` of the function at level `level` of the stack. (The first parameter or local variable has index 1, and so on, until the last active local variable.) The function returns **nil** if there is no local variable with the given index, and raises an error when called with a `level` out of range (you may call *`debug.getinfo`* to check whether the level is valid).

Variable names starting with `(´ (open parentheses) represent internal variables (loop control variables, temporaries, and C function locals).

debug.getmetatable (object)

Returns the metatable of the given `object` or **nil** if it does not have a metatable.

debug.getregistry ()

Returns the registry table (see B.3.6).

debug.getupvalue (func, up)

This function returns the name and the value of the upvalue with index `up` of the function `func`. The function returns **nil** if there is no upvalue with the given index.

debug.setfenv (object, table)

Sets the environment of the given `object` to the given `table`.

© ABNT 2011 - All rights reserved

debug.sethook (hook, mask [, count])

Sets the given function as a hook. The string `mask` and the number `count` describe when the hook will be called. The string mask may have the following characters, with the given meaning:

- **`"c"`**: the hook is called every time Lua calls a function;
- **`"r"`**: the hook is called every time Lua returns from a function;
- **`"l"`**: the hook is called every time Lua enters a new line of code.

With a `count` different from zero, the hook is called after every `count` instructions.

When called without arguments, *`debug.sethook`* turns off the hook.

When the hook is called, its first parameter is a string describing the event that has triggered its call: `"call"`, `"return"` (or `"tail return"`), `"line"`, and `"count"`. For line events, the hook also gets the new line number as its second parameter. Inside a hook, you may call `getinfo` with level 2 to get more information about the running function (level 0 is the `getinfo` function, and level 1 is the hook function), unless the event is `"tail return"`. In this case, Lua is only simulating the return, and a call to `getinfo` will return invalid data.

debug.setlocal (level, local, value)

This function assigns the value `value` to the local variable with index `local` of the function at level `level` of the stack. The function returns **nil** if there is no local variable with the given index, and raises an error when called with a `level` out of range (you may call `getinfo` to check whether the level is valid). Otherwise, it returns the name of the local variable.

debug.setmetatable (object, table)

Sets the metatable for the given `object` to the given `table` (which may be **nil**).

debug.setupvalue (func, up, value)

This function assigns the value `value` to the upvalue with index `up` of the function `func`. The function returns **nil** if there is no upvalue with the given index. Otherwise, it returns the name of the upvalue.

debug.traceback ([message])

Returns a string with a traceback of the call stack. An optional `message` string is appended at the beginning of the traceback. This function is typically used with `xpcall` to produce better error messages.

B.6 Lua stand-alone

Although Lua has been designed as an extension language, to be embedded in a host C program, it is also frequently used as a stand-alone language. An interpreter for Lua as a stand-alone language, called simply `lua`, is provided with the standard distribution. The stand-alone interpreter includes all standard libraries, including the debug library. Its usage is:

```
lua [options] [script [args]]
```

The options are:

- **`-e`** ***stat***: executes string *stat*;

© ABNT 2011 - All rights reserved

- `-l` ***mod***: "requires" *mod*;
- `-i`: enters interactive mode after running *script*;
- `-v`: prints version information;
- `--`: stops handling options;
- `-`: executes `stdin` as a file and stops handling options.

After handling its options, `lua` runs the given *script*, passing to it the given *args* as string arguments. When called without arguments, `lua` behaves as `lua -v -i` when the standard input (`stdin`) is a terminal, and as `lua -` otherwise.

Before running any argument, the interpreter checks for an environment variable `LUA_INIT`. If its format is *@filename*, then `lua` executes the file. Otherwise, `lua` executes the string itself.

All options are handled in order, except `-i`. For instance, an invocation like

```
$ lua -e'a=1' -e 'print(a)' script.lua
```

will first set `a` to 1, then print the value of `a` (which is `1´), and finally run the file `script.lua` with no arguments (here `$` is the shell prompt. Your prompt may be different).

Before starting to run the script, `lua` collects all arguments in the command line in a global table called `arg`. The script name is stored at index 0, the first argument after the script name goes to index 1, and so on. Any arguments before the script name (that is, the interpreter name plus the options) go to negative indices. For instance, in the call

```
$ lua -la b.lua t1 t2
```

the interpreter first runs the file `a.lua`, then creates a table

```
arg = { [-2] = "lua", [-1] = "-la",
        [0] = "b.lua",
        [1] = "t1", [2] = "t2" }
```

and finally runs the file `b.lua`. The script is called with `arg[1]`, `arg[2]`, ... as arguments; it may also access those arguments with the vararg expression `...´.

In interactive mode, if you write an incomplete statement, the interpreter waits for its completion by issuing a different prompt.

If the global variable `_PROMPT` contains a string, then its value is used as the prompt. Similarly, if the global variable `_PROMPT2` contains a string, its value is used as the secondary prompt (issued during incomplete statements). Therefore, both prompts may be changed directly on the command line. For instance,

```
$ lua -e"_PROMPT='myprompt> '" -i
```

The outer pair of quotes is for the shell, the inner pair is for Lua). The use of `-i` to enter interactive mode; otherwise, the program would just end silently right after the assignment to `_PROMPT`.

To allow the use of Lua as a script interpreter in Unix systems, the stand-alone interpreter skips the first line of a chunk if it starts with `#`. Therefore, Lua scripts may be made into executable programs by using `chmod +x` and the `#!` form, as in

```
#!/usr/local/bin/lua
```


© ABNT 2011 - All rights reserved

(Of course, the location of the Lua interpreter can be different in your machine. If `lua` is in your `PATH`, then

```
#!/usr/bin/env lua
```

is a more portable solution.)

B.7 Incompatibilities with the version 5.0

NOTE The incompatibilities that can be found when moving a program from Lua 5.0 to Lua 5.1 ares listed in this clause. You can avoid most of the incompatibilities compiling Lua with appropriate options (see file `luaconf.h`). However, all those compatibility options will be removed in the next version of Lua.

B.7.1 Changes in the language

These are the changes in the language introduced by Lua 5.1:

- the vararg system changed from the pseudo-argument `arg` with a table with the extra arguments to the vararg expression. (Option `LUA_COMPAT_VARARG` in `luaconf.h`);
- there was a subtle change in the scope of the implicit variables of the **for** statement and for the **repeat** statement;
- the long string/long comment syntax (`[[...]]`) does not allow nesting. You may use the new syntax (`[=[...]=]`) in those cases. (Option `LUA_COMPAT_LSTR` in `luaconf.h`).

B.7.2 Changes in the libraries

The changes in the libraries introduced by Lua 5.1 are the following:

- function string.gfind was renamed *string.gmatch* (see compile-time option LUA_COMPAT_GFIND in luaconf.h.);
- when *string.gsub* is called with a function as its third argument, whenever this function returns **nil** or **false** the replacement string is the whole match, instead of the empty string;
- function table.setn was deprecated. Function table.getn corresponds to the new length operator (#); use the operator instead of the function (see compile-time option LUA_COMPAT_GETN in luaconf.h);
- function loadlib was renamed *package.loadlib* (see compile-time option LUA_COMPAT_LOADLIB in luaconf.h);
- function math.mod was renamed math.fmod (see compile-time option LUA_COMPAT_MOD in luaconf.h);
- functions table.foreach and table.foreachi are deprecated. You can use a **for** loop with pairs or ipairs instead;
- there were substantial changes in function due to the new module system. However, the new behavior is mostly compatible with the old, but require gets the path from *package.path* instead of from LUA_PATH;
- function *collectgarbage* has different arguments. Function gcinfo is deprecated; use collectgarbage("count") instead.

© ABNT 2011 - All rights reserved

B.7.3 Changes in the API

These are the changes in the C API introduced by Lua 5.1:

- the luaopen_* functions (to open libraries) cannot be called directly, like a regular C function. They shall be called through Lua, like a Lua function;
- function lua_open was replaced by lua_newstate to allow the user to set a memory-allocation function. You can use luaL_newstate from the standard library to create a state with a standard allocation function (based on realloc);
- functions luaL_getn and luaL_setn (from the auxiliary library) are deprecated. Use lua_objlen instead of luaL_getn and nothing instead of luaL_setn;
- function luaL_openlib was replaced by luaL_register;

function luaL_checkudata now throws an error when the given value is not a userdata of the expected type. (In Lua 5.0 it returned NULL).

B.8 Complete syntax of Lua

The complete syntax of Lua in extended BNF is the following (it does not describe operator precedences):

```
chunk ::= {stat [`;´]} [laststat[`;´]]

block ::= chunk

stat ::=  varlist1 `=´ explist1  |

   functioncall  |

   do block end  |

   while exp do block end  |

   repeat block until exp  |

   if exp then block {elseif exp then block}[else block] end |

   for Name `=´ exp `,´ exp [`,´ exp] do block end  |

   for namelist in explist1 do block end  |

   function funcname funcbody  |

   local function Name funcbody  |

   local namelist [`=´ explist1]

 laststat ::= return [explist1]  |  break

funcname ::= Name {`.´ Name} [`:´ Name]

varlist1 ::= var {`,´ var}

var ::=  Name  |  prefixexp `[´ exp `]´  |  prefixexp `.´ Name
```


© ABNT 2011 - All rights reserved

```
namelist ::= Name {`,´ Name}
explist1 ::= {exp `,´} exp
exp ::= nil | false | true | Number | String | `...´ |
     function | prefixexp | tableconstructor |
     exp binop | exp | unop exp
prefixexp ::= var | functioncall | `(´ exp `)´
functioncall ::= prefixexp args | prefixexp `:´ Name args
args ::= `(´ [explist1] `)´ | tableconstructor | String
function ::= function funcbody
funcbody ::= `(´ [parlist1] `)´ block end
parlist1 ::= namelist [`,´ `...´] | `...´
tableconstructor ::= `{´ [fieldlist] `}´
fieldlist ::= field {fieldsep field} [fieldsep]
field ::= `[´ exp `]´ `=´ exp | Name `=´ exp | exp
fieldsep ::= `,´ | `;´
binop ::= `+´ | `-´ | `*´ | `/´ | `^´ | `%´ | `..´ |
      `<´ | `<=´ | `>´ | `>=´ | `==´ | `~=´ |
      and | or
```

unop ::= `-´ | not | `#´

© ABNT 2011 - All rights reserved

Anexo C
(informativo)

Connector base

This Connector Base may be imported by any NCL 3.0 document.

```
<!--

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/connectorBases/causalConnBase.ncl
Author: TeleMidia Laboratory
Revision: 19/09/2006

-->

<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="causalConnBase" xmlns="http://www.ncl.org.br/NCL3.0/CausalConnectorProfile">

<head>
<connectorBase>

<!-- OnBegin -->

<causalConnector id="onBeginStart">
  <simpleCondition role="onBegin"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onBeginStop">
  <simpleCondition role="onBegin"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onBeginPause">
  <simpleCondition role="onBegin"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onBeginResume">
  <simpleCondition role="onBegin"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onBeginSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnEnd -->

<causalConnector id="onEndStart">
  <simpleCondition role="onEnd"/>
  <simpleAction role="start"/>
```


© ABNT 2011 - All rights reserved

```
</causalConnector>

<causalConnector id="onEndStop">
  <simpleCondition role="onEnd"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onEndPause">
  <simpleCondition role="onEnd"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onEndResume">
  <simpleCondition role="onEnd"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onEndSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>


<!-- OnMouseSelection -->

<causalConnector id="onSelectionStart">
  <simpleCondition role="onSelection"/>
  <simpleAction role="start" />
</causalConnector>

<causalConnector id="onSelectionStop">
  <simpleCondition role="onSelection"/>
  <simpleAction role="stop" />
</causalConnector>

<causalConnector id="onSelectionPause">
  <simpleCondition role="onSelection"/>
  <simpleAction role="pause" />
</causalConnector>

<causalConnector id="onSelectionResume">
  <simpleCondition role="onSelection"/>
  <simpleAction role="resume" />
</causalConnector>

<causalConnector id="onSelectionSetVar">
  <connectorParam name="var" />
  <simpleCondition role="onSelection"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnKeySelection -->

<causalConnector id="onKeySelectionStart">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onKeySelectionStop">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="stop"/>
</causalConnector>
```

© ABNT 2011 - All rights reserved

```
<causalConnector id="onKeySelectionPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onKeySelectionResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onKeySelectionSetVar">
  <connectorParam name="keyCode"/>
  <connectorParam name="var"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnBeginAttribution -->

<causalConnector id="onBeginAttributionStart">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onBeginAttributionStop">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onBeginAttributionPause">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onBeginAttributionResume">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onBeginAttributionSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnEndAttribution -->

<causalConnector id="onEndAttributionStart">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onEndAttributionStop">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onEndAttributionPause">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="pause"/>
</causalConnector>
```


© ABNT 2011 - All rights reserved

```
<causalConnector id="onEndAttributionResume">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onEndAttributionSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>


<!-- OnBegin multiple actions -->

<causalConnector id="onBeginStartStop">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartResume">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopStart">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopResume">
  <simpleCondition role="onBegin"/>
```

© ABNT 2011 - All rights reserved

```
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnEnd multiple actions -->

<causalConnector id="onEndStartStop">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
```


© ABNT 2011 - All rights reserved

```
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartResume">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopStart">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopResume">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetStop">
```

© ABNT 2011 - All rights reserved

```
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stet" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnMouseSelection multiple actions -->

<causalConnector id="onSelectionStartStop">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopStart">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
```


© ABNT 2011 - All rights reserved

```
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stet" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>
```

© ABNT 2011 - All rights reserved

```
<!-- OnKeySelection multiple actions -->

<causalConnector id="onKeySelectionStartStop">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartSet">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopStart">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
```


© ABNT 2011 - All rights reserved

```
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopSet">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetStart">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetStop">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetPause">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetResume">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnBeginAttribution multiple actions -->

<causalConnector id="onBeginAttributionStartStop">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartPause">
```

© ABNT 2011 - All rights reserved

```
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartResume">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopStart">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopPause">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopResume">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>
```


© ABNT 2011 - All rights reserved

```
<causalConnector id="onBeginAttributionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnEndAttribution multiple actions -->

<causalConnector id="onEndAttributionStartStop">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartPause">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartResume">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopStart">
```

© ABNT 2011 - All rights reserved

```
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopPause">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopResume">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stet" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
```


© ABNT 2011 - All rights reserved

```
  </compoundAction>
</causalConnector>


<!--Miscellaneous-->

<causalConnector id="onKeySelectionStopResizePauseStart">
  <connectorParam name="width"/>
  <connectorParam name="height"/>
  <connectorParam name="left"/>
  <connectorParam name="top"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="setWidth" value="$width"/>
    <simpleAction role="setHeight" value="$height"/>
    <simpleAction role="setLeft" value="$left"/>
    <simpleAction role="setTop" value="$top"/>
    <simpleAction role="pause"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndResizeResume">
  <connectorParam name="left"/>
  <connectorParam name="top"/>
  <connectorParam name="width"/>
  <connectorParam name="height"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="setLeft" value="$left"/>
    <simpleAction role="setTop" value="$top"/>
    <simpleAction role="setWidth" value="$width"/>
    <simpleAction role="setHeight" value="$height"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopSetPauseStart">
  <connectorParam name="bounds"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$bounds"/>
    <simpleAction role="pause"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

</connectorBase>
</head>

</ncl>
```

© ABNT 2011 - All rights reserved

Bibliography

[1] ITU Recommendation J.201:2004, *Harmonization of declarative content format for interactive television applications*

[2] ARIB STD-B24:2004, *Data coding and transmission specifications for digital broadcasting*

[3] Cascading Style Sheets, Cascading Style Sheets, level 2, Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs. W3C Recommendation 12. Maio de 1998, disponínel em <http://www.w3.org/TR/REC-CSS2>

[4] DVB-HTML, Perrot P. DVB-HTML - An Optional Declarative Language within MHP 1.1, EBU Technical Review. 2001

[5] Namespaces in XML, Namespaces in XML, W3C Recommendation. Janeiro de 1999

[6] NCM Core, Soares L.F.G; Rodrigues R.F. Nested Context Model 3.0: Part 1 – NCM Core, Technical Report, Departamento de Informática PUC-Rio. Maio de 2005, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[7] NCL Digital TV Profiles, Soares L.F.G; Rodrigues R.F. Part 8 – NCL (Nested Context Language) Digital TV Profiles, Technical Report, Departamento de Informática PUC-Rio, No. 35/06. Outubro de 2006, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[8] NCL Live Editing Commands, Soares L.F.G; Rodrigues R.F; Costa, R.R.; Moreno, M.F. Part 9 – NCL Live Editing Commands. Technical Report, Departamento de Informática PUC-Rio, No. 36/06. Dezembro de 2006, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[9] NCL-Lua, Cerqueira, R,; Sant'Anna, F. Nested Context Model 3.0: Part 11 – Lua Scripting Language for NCL, Technical Report, Departamento de Informática PUC-Rio. Maio de 2007, ISSN: 0103-9741.

[10] NCL Main Profile, Soares L.F.G; Rodrigues R.F; Costa, R.R. Nested Context Model 3.0: Part 6 – NCL (Nested Context Language) Main Profile, Technical Report, Departamento de Informática PUC-Rio. Maio de 2005, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[11] RDF, Resource Description Framework (RDF) Model and Syntax Specification, Ora Lassila and Ralph R. Swick. W3C Recommendation. 22 de fevereiro de 1999. Disponível em <http://www.w3.org/TR/REC-rdf-syntax/>

[12] SMIL 2.1 Specification, SMIL 2.1 - *Synchronized Multimedia Integration Language – SMIL 2.1 Specification,* W3C Recommendation. Dezembro de 2005

[13] XHTML 1.0, XHTML™ 1.0 2º Edition - *Extensible HyperText Markup Language*, W3C Recommendation, Agosto de 2002

[14] Programming in Lua, Segunda Edição, de Roberto Ierusalimschy et al. Março de 2006, ISBN 85-903798-2-5.

[15] ACAP, Advanced Application Platform (ACAP), ATSC Standard: Document A/101. Agosto de 2005

© ABNT 2011 - All rights reserved