Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações

APRESENTAÇÃO

1) Este Projeto foi elaborado pela Comissão de Estudo Especial de Televisão Digital (ABNT/CEE-85), nas reuniões de:

15.12.2014	16.12.2014	17.12.2014
08.05.2015		---

2) Não tem valor normativo;

3) Aqueles que tiverem conhecimento de qualquer direito de patente devem apresentar esta informação em seus comentários, com documentação comprobatória;

4) Este Projeto de Norma será diagramado conforme as regras de editoração da ABNT quando de sua publicação como Norma Brasileira;

5) Tomaram parte na elaboração deste Projeto:

Participante Representante

Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações

Digital terrestrial television – Data coding and transmission specification for digital broadcasting – Parte2: Ginga-NCL for fixed and mobile receivers – XML application language for application coding

Prefácio

A Associação Brasileira de Normas Técnicas (ABNT) é o Foro Nacional de Normalização. As Normas Brasileiras, cujo conteúdo é de responsabilidade dos Comitês Brasileiros (ABNT/CB), dos Organismos de Normalização Setorial (ABNT/ONS) e das Comissões de Estudo Especiais (ABNT/CEE), são elaboradas por Comissões de Estudo (CE), formadas por representantes dos setores envolvidos, delas fazendo parte: produtores, consumidores e neutros (universidades, laboratórios e outros).

Os Documentos Técnicos ABNT são elaborados conforme as regras da Diretiva ABNT, Parte 2.

Esta Norma é baseada nos trabalhos do Fórum do Sistema Brasileiro de Televisão Digital Terrestre, conforme estabelecido no Decreto Presidencial nº 5.820, de 29.06.2006.

O Escopo desta Norma Brasileira em inglês é o seguinte:

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.

Emenda

Página 38, Seção 7.2.4, Tabela 12

Inserir após a linha 8 (listada abaixo) da Tabela

system.audioType	Tipo de áudio do dispositivo de exibição, quando uma classe não for definida	“mono” | “stereo” | “5.1”

as linhas

system.persistent	Quantidade de memória interna, não volátil, disponível (em MByte)	real
system.makerId	Identificador do fabricante do receptor (de acordo com a Tabela 75 da ABNT NBR 15608-3)	string
system.modelId	Identificador do modelo do receptor (definido por cada fabricante)	string
system.serialNumber	Número de série do receptor (definido por cada fabricante)	string
system.versionId	Identificador da versão do software do receptor (definido por cada fabricante)	string
system.macAddress	Endereço MAC (IEEE 802) da interface de rede ativa do receptor	string (hh:hh:hh:hh:hh:hh, onde h é um inteiro hexadecimal)

Página 50, Seção 7.2.8, 1º. parágrafo da página

Substituir:

— evento de atribuição, que é definido pela atribuição de um valor a uma propriedade de um nó (representado por um elemento <media>, <body>, <context> ou <switch>), que deve obrigatoriamente ser declarado em um elemento <property>, filho do nó; e

Por:

— evento de atribuição, que é definido pela atribuição de um valor a uma propriedade de um nó (representado por um elemento <media>, <body>, ou <context>), que deve obrigatoriamente ser declarado em um elemento <property>, filho do nó; e

---

Página 58, Seção 7.2.10

Inserir após o primeiro parágrafo da seção:

NOTA O attributo *explicitDur* deve ser especificado de acordo com uma das seguintes sintaxes:
i) Horas":"Minutos":"Segundos"."Fração, onde Horas é um inteiro no intervalo [0, 23]; Minutos é um inteiro no intervalo [0,59]; Segundos é um inteiro no intervalo [0,59]; e Fração é um inteiro positivo.
ii) Segundos"s", onde Segundos é um número real positivo.

Página 58, Seção 7.2.11, 5º. parágrafo da seção

Substituir:

— o elemento <descriptorBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a um outro documento NCL contendo a base de descritores a ser importada (na verdade seus elementos-filhos) e aninhada. Quando uma base de descritores é importada, a base de regiões e a base de regras, quando existentes no documento importado, são também automaticamente importadas para as bases de regiões e de regras do documento correspondente que realiza a importação;

Por:

— o elemento <descriptorBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a um outro documento NCL contendo a base de descritores a ser importada (na verdade seus elementos-filhos) e aninhada. Quando uma base de descritores é importada, a base de regiões, a base de transições e a base de regras, quando existentes no documento importado, são também automaticamente importadas para as bases de regiões, de transições e de regras do documento correspondente que realiza a importação;

Página 58, Seção 7.2.11, 9º. parágrafo da seção

Substituir:

— o elemento <regionBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a um outro documento NCL contendo a base da regiões a ser importada (na verdade seus elementos-filhos) e aninhada. Como o documento referido pode ter mais de uma base de regiões, a base a ser importada é identificada é identificada pela atribuição de seu identificador ao atributo *baseId*. Embora NCL defina seu modelo de leiaute, nada impede que um documento NCL utilize outros modelos de leiaute, desde que eles definam regiões onde os objetos podem ser apresentados, como, por exemplo, modelos de leiaute SMIL 2.0 [13]. Ao importar uma <regionBase>, um atributo opcional denominado *region* pode ser especificado, dentro de um elemento <importBase>. Quando presente, o atributo deve

obrigatoriamente identificar o *id* de um elemento <region> declarado no elemento <regionBase> do documento hospedeiro (documento que fez a operação de importação). Como conseqüência, todos os elementos <region>, filhos do <regionBase> importados, devem obrigatoriamente ser considerados elementos <region> filhos da região referida pelo atributo *region* do <importBase>. Se não especificado, os elementos <region>, filhos do <regionBase> importado, devem obrigatoriamente ser considerados filhos diretos do elemento <regionBase> do documento hospedeiro.

Por:

— o elemento <regionBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a um outro documento NCL contendo a base da regiões a ser importada (na verdade seus elementos-filhos) e aninhada. Como o documento referido pode ter mais de uma base de regiões, a base a ser importada é identificada é identificada pela atribuição de seu identificador ao atributo *baseId*. Embora NCL defina seu modelo de leiaute, nada impede que um documento NCL utilize outros modelos de leiaute, desde que eles definam regiões onde os objetos podem ser apresentados, como, por exemplo, modelos de leiaute SMIL 2.0 [13]. Ao importar uma <regionBase>, um atributo opcional denominado *region* pode ser especificado, dentro de um elemento <importBase>. Quando presente, o atributo deve obrigatoriamente identificar o *id* de um elemento <region> declarado em um elemento <regionBase> do documento hospedeiro (documento que fez a operação de importação). Como conseqüência, todos os elementos <region>, filhos do <regionBase> importados, devem obrigatoriamente ser considerados elementos <region> filhos da região referida pelo atributo *region* do <importBase>. Se não especificado, os elementos <region>, filhos do <regionBase> importado, devem obrigatoriamente ser considerados filhos diretos do elemento <regionBase> do documento hospedeiro.

Página 96, Seção 9.1, parágrafo 1 a 3 da seção

Substituir:

O núcleo da máquina de apresentação Ginga-NCL é composto pelo formatador NCL e seu módulo Gerenciador de Base Privada.

O formatador NCL é responsável por receber um documento NCL e controlar sua apresentação, tentando garantir que as relações especificadas entre os objetos de mídia sejam respeitadas. O formatador lida com documentos NCL que são coletados dentro de uma estrutura de dados conhecida como base privada. Ginga associa uma base privada a um canal de televisão. Os documentos NCL em uma base privada podem ser iniciados, pausados, retomados, parados e podem referir-se uns aos outros.

O Gerenciador de Base Privada é responsável por receber comandos de edição de documentos NCL e pela edição dos documentos NCL ativos (documentos sendo apresentados).

Por:

No Ginga, aplicações são coletadas dentro de uma estrutura de dados conhecida como base privada. As aplicações em uma base privada podem ser iniciadas, pausadas, retomadas, paradas e podem referir-se umas às outras.

O núcleo da máquina de apresentação Ginga-NCL é composto pelo formatador NCL. O formatador NCL é responsável por receber um documento NCL e controlar sua apresentação, tentando garantir que as relações especificadas entre os objetos de mídia sejam respeitadas. O formatador lida com documentos NCL que são coletados dentro de uma base privada. Ginga associa uma ou mais bases privadas a cada canal de televisão.

O Gerenciador de Base Privada é o módulo Ginga responsável por receber comandos para manipulação das bases privadas e das aplicações nelas contidas. Em particular, por receber comandos de edição de documentos NCL e pela edição dos documentos NCL ativos (documentos sendo apresentados).

Página 97, Seção 9.1, parágrafo 3 da página

Substituir:

Os arquivos de documento NCL e os conteúdos de objeto de mídia NCL são organizados em estruturas de sistemas de arquivos. Os parâmetros de comando de edição, baseados em XML, podem ser diretamente transportados no *payload* de um descritor de evento ou, alternativamente, organizados em estruturas de sistema de arquivos a serem transportadas no canal de difusão de dados, ou ainda serem recebidas pelo canal de interatividade.

Por:

Os arquivos de documento NCL e os conteúdos de objetos de mídia NCL são organizados em estruturas de sistemas de arquivos. Os parâmetros de comando de edição, baseados em XML, podem ser diretamente transportados no *payload* de um descritor de evento ou, alternativamente, organizados em estruturas de sistema de arquivos a serem transportadas no canal de difusão de dados, ou ainda serem recebidas pelo canal de interatividade.

Página 98, Seção 9.1, 1º. parágrafo da página

Substituir:

A Tabela 56 mostra as *strings* de comando e, cercados por parênteses, os parâmetros transportados como conteúdo *payload* do descritor de evento *nclEditingCommand*. Os comandos são divididos em três grupos: o primeiro para operação da base privada (para abrir, ativar, desativar, fechar e salvar bases privadas); o segundo para manipulação de documentos (para adicionar, remover e salvar um documento em uma base privada aberta e para iniciar, pausar, retomar e parar apresentações de documentos em uma base privada ativa); e a última para manipular entidades NCL em uma base priveda aberta. Para cada entidade NCL, foram definidos os comandos *add* e *remove*. Se uma entidade já existir, o comando *add* tem a semântica de atualização (alteração).

Por:

A Tabela 56 mostra as *strings* de comando e, cercados por parênteses, os parâmetros transportados como conteúdo *payload* do descritor de evento *nclEditingCommand*. Os comandos são divididos em três grupos: o primeiro para operação de bases privadas (para abrir, ativar, desativar, fechar e salvar bases privadas); o segundo para manipulação de aplicações (para adicionar, remover e salvar uma aplicação em uma base privada aberta e para iniciar, pausar, retomar e parar apresentações de aplicações em uma base privada ativa); e a última para manipular entidades NCL em uma base privada aberta. Para cada entidade NCL, foram definidos os comandos *add* e *remove*. Se uma entidade já existir, o comando *add* tem a semântica de atualização (alteração).

Página 98, Seção 9.1, Tabela 56

Substituir as linhas 2 a 7 da tabela

openBase (baseId, location)	0x00	Abre uma base privada existente localizada pelo parâmetro location. Se a base privada não existir ou se o parâmetro location não for informado, uma nova base é criada com o identificador baseId. O parâmetro location deve obrigatoriamente especificar o dispositivo de armazenamento no ambiente do receptor e o caminho onde está a base a ser aberta.
activateBase (baseId)	0x01	Ativa uma base privada aberta. Todas as aplicações ficam então aptas a seres iniciadas.
deactivateBase (baseId)	0x02	Desativa uma base privada aberta. Todas as aplicações devem ser terminadas.
saveBase (baseId, location)	0x03	Salva todo o conteúdo da base privada em um dispositivo de armazenamento persistente (se disponível). O parâmetro location deve obrigatoriamente especificar o dispositivo e

		caminho para salvar a base
closeBase (baseId)	0x04	Fecha a base privada aberta e descarta todo o conteúdo da base privada
addDocument (baseId, {uri, id}+)	0x05	Adiciona um documento NCL a uma base privada aberta. Os arquivos do documento NCL podem ser: i) enviados sem solicitação pela rede de difusão de dados; nesse caso, o par {uri, id} é usado para relacionar um conjunto de caminhos de arquivos especificados no documento NCL com suas respectivas localizações no sistema de transporte (veja exemplos na Seção 12); NOTA Os conjuntos de pares de referência devem obrigatoriamente ser suficientes para que o middleware possa mapear qualquer referência a arquivos presentes na especificação do documento NCL na sua localização concreta na memória do dispositivo receptor. ii) recebidos pelo canal de interatividade sob demanda, ou já serem residentes no receptor; para esses arquivos, nenhum par {uri, id} necessita ser enviado, exceto o par {uri, "null"} associado ao documento NCL que deverá ser adicionado na base baseId, se o documento NCL não for recebido sem solicitação (pushed file).

Por:

openBase (baseId, location)	0x00	Abre uma base privada existente. Se a base privada não existir uma nova base é criada na PBDS. O parâmetro location especifica a localização da base a ser aberta na PBDS. O parâmetro baseId identifica a base privada, univocamente. Como, usualmente (e esse é o caso no ISDB-T), essa identificação é feita por meio do caminho de aninhamento de bases na PBDS, caminho no qual a base a ser aberta deve estar inserida como último aninhamento, o parâmetro location pode ser omitido, quando então é inferido a partir do parâmetro *baseId*.
activateBase (baseId)	0x01	Ativa uma base privada aberta. Todas as aplicações ficam então aptas a seres iniciadas.
deactivateBase (baseId)	0x02	Desativa uma base privada aberta. Todas as aplicações devem ser terminadas.
saveBase (baseId, location)	0x03	Torna persistente todas as aplicações da base privada em um dispositivo de armazenamento não volátil (se disponível). O parâmetro location especifica o dispositivo e caminho para salvar o conteúdo da base. Alternativamente, location pode receber como valor *extMem,* identificando um dispositivo de armazenamento externo padrão (default) definido pelo receptor, ou o valor *intMem,* significando que a base se tornará persistente na memória não volátil interna do receptor. Se salva em dispositivo de armazenamento externo, a integridade das aplicações da base deve ser garantida pelo middeware para que as aplicações possam ser executadas. Para que uma alicação seja tornada persistente, é necessário que todos os arquivos transmitidos sem solicitação (pushed data) já estejam armazenados na PBDS.
closeBase (baseId)	0x04	Fecha a base privada desativada e descarta todo seu conteúdo não persistente.
removeBase (baseId)	0x35	Remove a base privada fechada da PBDS.
addDocument (baseId, {uri, id}+)	0x05	Adiciona um documento NCL a uma base privada aberta. Os arquivos do documento NCL podem ser: i) enviados sem solicitação pela rede de difusão de dados; nesse caso, o par {uri, id} é usado para relacionar um conjunto de caminhos de arquivos especificados no documento NCL com

		suas respectivas localizações no sistema de transporte (veja exemplos na Seção 12); NOTA Os conjuntos de pares de referência devem obrigatoriamente ser suficientes para que o middleware possa mapear qualquer referência a arquivos presentes na especificação do documento NCL na sua localização concreta na memória do dispositivo receptor. ii) recebidos pelo canal de interatividade sob demanda, ou já serem residentes no receptor; para esses arquivos, nenhum par {uri, id} necessita ser enviado, exceto o par {uri, "null"} associado ao documento NCL que deverá ser adicionado na base baseId, se o documento NCL não for recebido sem solicitação (pushed file).
addApplication (baseId, {uri, id}+, applicationId, {minResourceURI}+, [NVStorageURI]+)	0x36	Adiciona uma aplicação NCL a uma base privada aberta. Os arquivos da aplicação NCL podem ser: i) enviados sem solicitação pela rede de difusão de dados; nesse caso, o par {uri, id} é usado para relacionar um conjunto de caminhos de arquivos especificados no documento NCL com suas respectivas localizações no sistema de transporte (veja exemplos na Seção 12); NOTA Os conjuntos de pares de referência devem obrigatoriamente ser suficientes para que o middleware possa mapear qualquer referência a arquivos presentes na especificação do documento NCL na sua localização concreta na memória do dispositivo receptor. ii) recebidos pelo canal de interatividade sob demanda, ou já serem residentes no receptor; para esses arquivos, nenhum par {uri, id} necessita ser enviado, exceto o par {uri, "null"} associado ao documento NCL que deverá ser adicionado na base baseId, se o documento NCL não for recebido sem solicitação (pushed file). O parâmetro applicationId corresponde ao *id* do elemento <ncl> do documento NCL que é o ponto de entrada da aplicação. A lista de parâmetros minResourceURI contém a lista de URIs (separados por vírgula) de arquivos que já devem estar carregados em memória antes de iniciar a apresentação da aplicação. O parâmetro pode ser omitido, significando que a aplicação pode ser iniciada independentemente dos arquivos carregados em memória, ou receber o valor [all], significando que todos os arquivos transmitidos sem solicitação (pushed data) devem estar disponíveis na memória. A lista de parâmetros NVStorageURI contém a lista de URIs (separados por vírgula) de arquivos que se recomenda o armazenamento em memória não volátil. O parâmetro pode ser omitido, significando que não há qualquer recomendação, ou receber o valor [all], significando que é recomendado que todos os arquivos transmitidos sem solicitação (pushed data) sejam armazenados em memória não volátil.
removeApplication (baseId, applicationId)	0x37	Remove uma aplicação NCL de uma base privada aberta. A aplicação deve ser parada caso ainda esteja em execução.

Página 99, Seção 9.1, Tabela 56

Substituir a linha 13 da tabela

saveDocument (baseId, documented, location)	0x2E	Salva um documento NCL de uma base privada aberta em um dispositivo de armazenamento persistente (se disponível). O parâmetro location deve especificar o dispositivo e o caminho no dispositivo onde o documento será salvo. Se o documento NCL estiver sendo exibido, ele deve primeiro ser parado (todos os

		eventos no estado *occurring* devem ser parados).

Por:

saveApplication (baseId, applicationId, location)	0x3C	Torna persistente uma aplicação NCL de uma base privada aberta em um dispositivo de armazenamento não volátil (se disponível). O parâmetro location especifica o dispositivo e caminho para tornar persistente a aplicação. Alternativamente, location pode receber como valor *extMem,* identificando um dispositivo de armazenamento externo padrão (default) definido pelo receptor, ou o valor *intMem,* significando que a base se tornará persistente na memória não volátil interna do receptor. Se salva em dispositivo de armazenamento externo, a integridade da aplicação deve ser garantida pelo middeware para que possa ser executada. Se o documento NCL estiver sendo exibido, ele deve primeiro ser parado (todos os eventos no estado *occurring* devem ser parados). Para que uma alicação seja tornada persistente,, é necessário que todos os arquivos transmitidos sem solicitação (pushed data) já estejam armazenados na PBDS.
saveDocument (baseId, documentId, location)	0x2E	Salva um documento NCL de uma base privada aberta em um dispositivo de armazenamento persistente (se disponível). O parâmetro location deve especificar o dispositivo e o caminho no dispositivo onde o documento será salvo. Se o documento NCL estiver sendo exibido, ele deve primeiro ser parado (todos os eventos no estado *occurring* devem ser parados).

Página 102, Seção 9.1, 1º. E 2º. Parágrafos após a Tabela 56

Substituir:

Os receptores que somente implementam o perfil NCL DTV Básico podem não lidar com os seguintes comandos: *pauseDocument*, *resumeDocument*, *addTransition*, *removeTransition*, *addTransitionBase* e *removeTransitionBase*.

Ginga associa pelo menos uma base privada a cada canal de televisão. Quando um canal é sintonizado, sua base privada correspondente é aberta e ativada pelo Gerenciador da Base Privada; outras bases privadas devem ser desativadas. Por razões de segurança, apenas uma única base privada pode estar ativa por vez. O modo mais simples e restritivo de gerenciar bases privadas é ter uma única base privada aberta por vez. Assim, se o usuário mudar o canal selecionado, recomenda-se que a base privada atual seja fechada. Nesse caso, o comando *openBase* é sempre seguido pelo comando *activeBase* e o comando *deactiveBase* nunca é usado. Contudo, o número de bases privadas que podem ser mantidas em aberto é uma decisão de implementação do middleware.

Por:

Os receptores que somente implementam o perfil NCL DTV Básico podem não lidar com os seguintes comandos: *pauseDocument*, *resumeDocument*, *addTransition*, *removeTransition*, *addTransitionBase* e *removeTransitionBase*.

O Ginga associa pelo menos uma base privada a cada canal de TV (conjunto de serviços): chamada base privada padrão (*default*) do canal. Quando um canal é sintonizado, sua base privada default é aberta e ativada pelo Gerenciador de Base privada. As outras bases privadas devem obrigatoriamente ser desativadas. Através do canal sintonizado outras bases privadas filhas (aninhadas) podem ser abertas (ou criadas), mas no máximo uma pode ser associada com cada serviço do canal sintonizado. Quando o canal tem apenas um serviço, como é o caso da recepção one-seg, uma e somente uma base privada é associada a cada canal de TV (a base privada *default* do canal). Nesse caso, qualquer comando de edição que venha pelo canal de difusão para abrir uma base privada associada a um serviço deve ser ignorado.

NOTA A base privada associada ao canal de TV deve obrigatoriamente ter o identificador (parâmetro baseId)

igual ao “valor do campo network_id da NIT (table_id 0x40)”. As outras possíveis bases privadas associadas a um serviço do canal de TV devem obrigatoriamente ter seu identificador (baseId parameter) igual ao “valor do campo network_id da NIT.valor do campo program_number da PMT”. O valor do *program_number* é aquele associado ao serviço correspondente.

As aplicações residentes são gerenciadas em uma base privada específica. Da mesma forma, aplicações instaladas são gerenciadas em uma base privada específica.

Por razões de segurança, pode-se ativar apenas uma base privada por vez, dentre aquelas controladas por meio do canal de transmissão sintonizado. A forma mais simples e restrita para gerenciar bases privadas é ter apenas uma base privada aberta por vez, dentre aquelas controladas por meio do canal de transmissão sintonizado. Assim, se o usuário mudar o canal selecionado, recomenda-se que a base privada atual seja fechada. Nesse caso, o comando *openBase* é sempre seguido pelo comando *activateBase* e o comando *deactivateBase* nunca é usado. No entanto, o número de bases privadas que podem ser mantidas abertas é uma decisão da implementação específica do *middleware*.

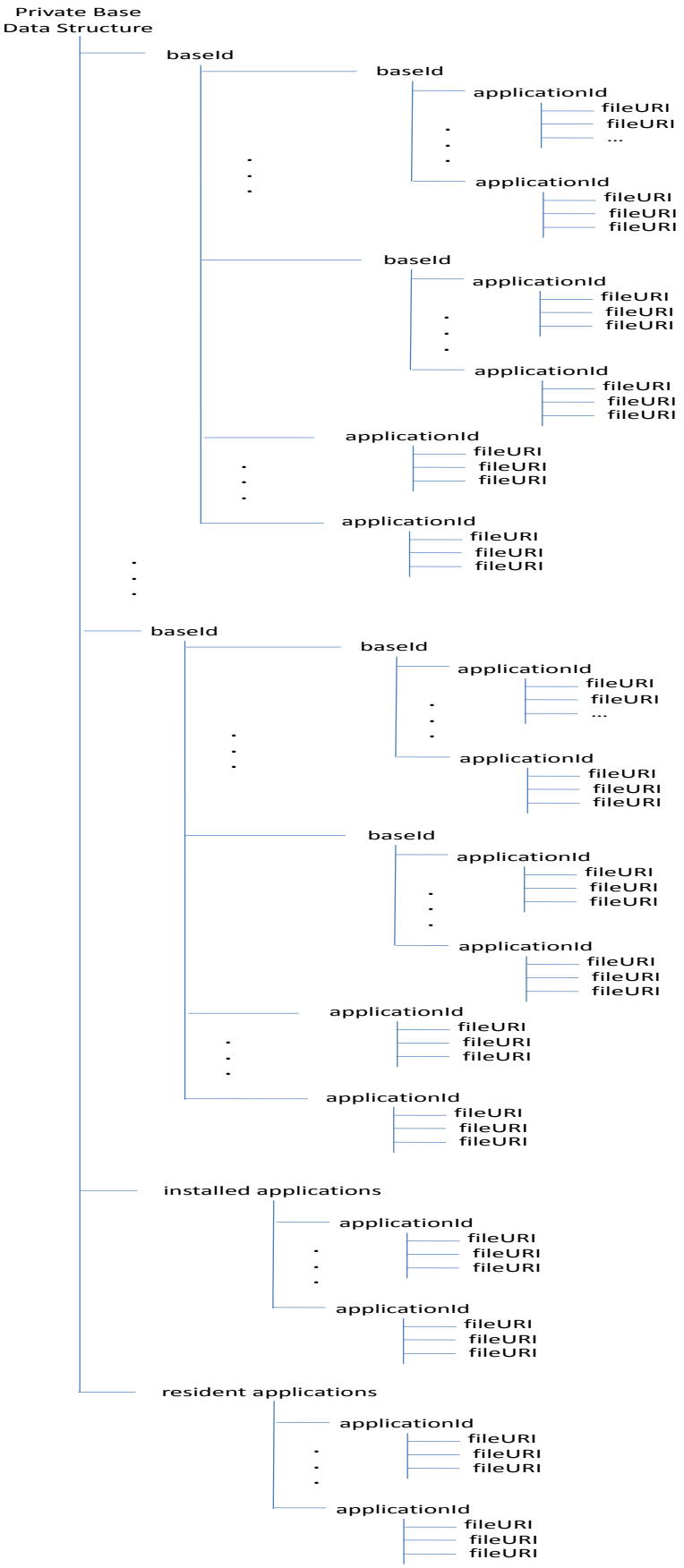
NOTA Apenas bases privadas criadas por comandos de edição enviados pelo canal de difusão sintonizado e a base privada default do canal de TV podem ser controladas por comandos de edição provenientes do mesmo canal sintonizado.

NOTA Os eventos NCLua e NCLet (eventos de comando de edição NCL) gerados por objetos imperativos de aplicações operando em uma base privada associada a um canal de TV podem controlar apenas as bases privadas associadas a esse canal.

Toda estrutura de dados de bases privadas é mantida em memória volátil ou não volátil. Bases privadas e aplicações se tornam persistentes quando são explicitamente salvas.

NOTA: Salienta-se a diferença entre persistência e tipo de memória. Uma aplicação quando persistente deve ser armazenada em memória não volátil. No entanto, enquanto não-persistente, pode residir em memória volátil, não-volátil, ou em ambas.

A estrutura de dados de base privadas (PBDS) é resumida pelo Ginga em uma estrutura em árvore, cujo acesso é oferecido para leitura pelas aplicações. Cabe a todos os subsistemas do Ginga e suas extensões, incluindo o subsistema Ginga-NCL e a extensão Ginga-J, a manutenção dessa estrutura em árvore espelhando, de forma resumida, a estrutura de dados de base privadas. Um exemplo da estrutura em árvore é apresentada a seguir:


Private Base
Data Structure
baseId
baseId
applicationId
fileURI
fileURI
...
applicationId
fileURI
fileURI
fileURI
baseId
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI
baseId
baseId
applicationId
fileURI
fileURI
...
applicationId
fileURI
fileURI
fileURI
baseId
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI
installed applications
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI
resident applications
applicationId
fileURI
fileURI
fileURI
applicationId
fileURI
fileURI
fileURI

As aplicações residentes e as instaladas são gerenciadas em uma base específica a cada tipo, conforme apresentado na figura exemplo.

Todo nó da árvore possui um conjunto de informações associadas. O nó da árvore *baseId* (identificado pelo valor do campo network_id da NIT ou o "valor do campo network_id da NIT.valor do campo program_number da PMT") tem como informação associada obrigatória se o armazenamento da base privada está persistente ou não (atributo *persistent*), bem como o estado atual da base (atributo *state* igual a aberta, ativa, ou fechada). O nó da árvore *applicationId* (identificado pelo *id* da aplicação - no caso de uma aplicação NCL, pelo *id* do elemento <ncl> do documento NCL que é o ponto de entrada da aplicação) tem como informações associadas obrigatórias a URI do código de ponto de entrada da aplicação (atributo *entryPoint*), o conjunto de recursos mínimos (URIs de arquivos que compõem a aplicação) que devem estar armazenado na PBDS para que a aplicacão seja iniciada (atributo *minResources*), se o armazenamento da aplicação está persistente ou não (atributo *persistent*) e se o armazenamento dos conteúdos da aplicação recebidos sem solicitação estão completos (atributo *completion*). Os nós da árvore *fileURI* (identificados pelo nome do arquivo correspondente) correspondem ao arquivos de código da aplicação (arquivo de código principal que é o ponto de entrada da aplicação e demais arquivos de código), arquivos de recursos da aplicação (arquivos de conteúdo de dados (conteúdos monomídia e multimídia) utilizados pela aplicação durante sua execução e que estão armazenados na PBDS, e arquivos de metadados da aplicação que estão armazenados na PBDS. Os nós da árvore *fileURI* têm como informação a localização de armazenamento na PBDS do arquivo correspondente, se o armazenamento é total ou parcial e quanto falta para o término do armazenamento, se o armazenamento for parcial (atributo *completion*).

Página 102, Seção 9.1,

Inserir após *3º. parágrafo da seção, descrito abaixo*

Os comandos *add* têm entidades NCL como seus argumentos (parâmetros de comando baseados em XML). Se a entidade especificada já existe ou não, a consistência do documento deve obrigatoriamente ser mantida pelo formatador NCL, no sentido de que todos os atributos de identidade classificados como obrigatórios devem obrigatoriamente ser definidos. As entidades são definidas utilizando uma notação sintática idêntica àquela usada pelos esquemas NCL, com exceção do comando *addInterface*: o atributo *begin* de um elemento <area> pode receber o valor "now", especificando o NPT atual do *nodeId*, que deve obrigatoriamente ser o vídeo principal MPEG sendo reproduzido pelo decodificador de *hardware*.

O texto a seguir:

Operações para o controle do ciclo de vida das aplicações via tabelas AIT podem ser usadas para manipulação de aplicações NCL e Java em uma base privada. Cada tabela contém dois laços (*loops*) de nível superior. O primeiro deles é o laço comum, que contém um conjunto de descritores que se referem a todas as aplicações sinalizadas naquela AIT particular, ou que se aplicam ao serviço como um todo. O outro laço de nível superior é o laço de aplicações, que descreve as aplicações sinalizadas por aquela instância de AIT. Cada iteração no laço de aplicações descreve uma aplicação (ou seja, representa uma linha na tabela de aplicações sinalizadas). Dentre essas informações, está o código de controle da aplicação que, no caso de aplicações NCL, deve obrigatoriamente adotar um dos valores mostrados na tabela a seguir.

Controle do ciclo de vida de aplicações NCL usando AIT (campo AIT: application_type ="0x0009")

campo AIT: **application_control_code**	**Descrição**	***nclEditingCommand* equivalente**	**Restrição**
AUTOSTART (0x01)	A aplicação é iniciada automaticamente quando o serviço for selecionado.	addApplication (baseId, {uri, id}, applicationId); seguido de startDocument (baseId,	a) A base privada é identificada pelo "valor do campo network_id da NIT", ou o "valor do campo program_number da

		documentId, interfaceId, 0, null, null) para adicionar (se já não estiver adicionada) e iniciar uma aplicação na base privada associada com o serviço sintonizado	PMT que refere à AIT"; b) O atributo id do elemento <ncl>, o parâmetro applicationId do comando addApplication e o parâmetro documentId do comando startDocument devem receber o mesmo valor do campo application_id da estrutura application_identifier da AIT; c) Para o comando addApplication equivalente: Há apenas um parâmetro uri que deve obrigatoriamente receber o valor definido no campo base_directory do ginga_ application_location_descriptor. Há apenas um parâmetro de id que deve obrigatoriamente receber o valor IOR correspondente ao documento definido no campo initial_entity do ginga_ application_location_descriptor. d) Para o comando addApplication equivalente: Os parâmetros {minResourceURI} e [NVStorageURI] são omitidos. e) A aplicação NCL deve obrigatoriamente ser iniciada a partir de todas as interfaces do elemento <body>; f) A aplicação NCL deve obrigatoriamente ser iniciada imediatamente, já que não pode ter seu tempo inicial se referindo a um valor NPT, que supõe-se ser

			"do it now"; g) A aplicação NCL deve obrigatoriamente ser iniciada do seu começo, já que o parâmetro *offset* é assumido como "0".
PRESENT (0x02)	A aplicação não é iniciada automaticamente. Ela é colocada em uma lista de aplicações disponíveis no receptor, podendo ser selecionada por um usuário para ser então iniciada.	Não existe comando de edição NCL correspondente. Aplicações NCL com tal comportamento só se sinalizadas pela AIT.	A aplicação não é iniciada automaticamente. Ela é colocada em uma lista de aplicações disponíveis no receptor, podendo ser selecionada por um usuário para ser então iniciada.
DESTROY (0x03)	A aplicação é interrompida	stopDocument (baseId, documentId) para interromper uma aplicação na base privada associada ao serviço de TV sintonizado ou na base privada *default* associada com o canal de TV sintonizado. O parâmetro baseId deve obrigatoriamente identificar a base privada associada ao serviço ou a base privada *default* associada ao canal de TV sintonizado	a) O parâmetro baseId no comando de edição NCL deve receber "o valor do campo network_id da NIT" ou "o valor do campo program_number da PMT que refere a AIT"; b) O atributo *id* do elemento <ncl> e o parâmetro documentId no comando de edição NCL deve obrigatoriamente receber o mesmo valor do campo application_id da estrutura application_identifier da AIT; c) A aplicação NCL deve obrigatoriamente ser imediatamente interrompida, já que não pode ter seu tempo final se referindo a um valor NPT, que é assumido como sendo "do it now"
KILL (0x04)	A aplicação é interrompida e removida do receptor	stopDocument (baseId, documentId) seguido de removeApplication(baseId , applicationId) para interromper uma aplicação, na base privada associada ao serviço sintonizado ou na base privada *default* associada com o canal de TV sintonizado, e então	a) O parâmetro baseId nos comandos de edição NCL deve receber "o valor do campo network_id da NIT" ou "o valor do campo program_number da PMT que refere a AIT"; b) O atributo *id* do elemento <ncl>, o parâmetro applicationId no comando remove Application e o parâmetro documentId no comando

		removê-la da base privada. O parâmetro baseId deve obrigatoriamente identificar a base privada associada ao serviço ou a base privada *default* associada ao canal de TV sintonizado	stopDocument devem obrigatoriamente receber o mesmo valor do campo application_id da estrutura application_identifier da AIT; c) A aplicação NCL deve obrigatoriamente ser imediatamente interrompida, já que não pode ter seu tempo final se referindo a um valor NPT, que é assumido como sendo "do it now" d) Uma aplicação NCL não pode ser reiniciada após de ter sido interrompida pelo controle KILL, sem ter todo o seu processo de sinalização refeito.
PREFETCH (0x05)	O exibidor NCL é preparado. O gerente de base privada deve obrigatoriamente aguardar o comando addDocument (baseId, {uri, id}+) para adicionar a aplicação na base privada baseId; e o comando startDocument (baseId, DocumentId, interfaceId, offset, nptBaseId, nptTrigger) para disparar a aplicação.		a) O parâmetro baseId nos comandos de edição NCL deve receber "o valor do campo network_id do NIT" ou "o valor do campo program_number da PMT que refere a AIT".
REMOTE (0x06)	A aplicação remota só estará disponível para ser executada após a seleção do serviço. .	addApplication (baseId, {uri, null}, applicationId); seguido de startDocument (baseId, documentId, interfaceId, 0, null, null) para adicionar (se já não estiver adicionada) e iniciar uma aplicação na base privada associada com o serviço sintonizado	a) A base privada é identificada pelo "valor do campo network_id da NIT", ou o "valor do campo program_number da PMT que refere à AIT"; b) O atributo id do elemento <ncl>, o parâmetro applicationId do comando addApplication e o parâmetro documentId do comando startDocument devem receber o mesmo valor do campo application_id da estrutura application_identifier da AIT; c) Para o comando

			addApplication equivalente: Há apenas um parâmetro uri que deve obrigatoriamente receber o valor definido no descritor correspondente da AIT que identifica a aplicação remota. Há apenas um parâmetro de id que deve obrigatoriamente receber o valor “null”. d) Para o comando addApplication equivalente: Os parâmetros {minResourceURI} e [NVStorageURI] são omitidos. e) A aplicação NCL deve obrigatoriamente ser iniciada a partir de todas as interfaces do elemento <body>; f) A aplicação NCL deve obrigatoriamente ser iniciada imediatamente, já que não pode ter seu tempo inicial se referindo a um valor NPT, que supõe-se ser "do it now"; A aplicação NCL deve obrigatoriamente ser iniciada do seu começo, já que o parâmetro *offset* é assumido como "0".
UNBOUND (0x07)	A aplicação pode ser armazenada de forma persistente sob comando do usuário. A aplicação não é iniciada automaticamente. Ela é colocada em uma lista de aplicações disponíveis no receptor, podendo ser selecionada por um usuário para então ser iniciada.	addApplication (baseId, {uri, id}, applicationId, [all]); seguido de saveApplication (baseId, applicationId, intMem) quando o usuário decidir tornar persistente a aplicação. seguido de startDocument (como em AUTOSTART)	a) A base privada é identificada pelo “valor do campo network_id da NIT”, ou o “valor do campo program_number da PMT que refere à AIT”; b) O atributo id do elemento <ncl> e o parâmetro applicationId devem receber o mesmo valor do campo application_id da estrutura application_identifier da AIT; c) Para o comando

		quando o usuário decidir por sua execução	addApplication equivalente: Há apenas um parâmetro uri que deve obrigatoriamente receber o valor definido no campo base_directory do ginga_ application_location_descriptor. Há apenas um parâmetro de id que deve obrigatoriamente receber o valor IOR correspondente ao documento definido no campo initial_entity do ginga_ application_location_descriptor. d) Para o comando addApplication equivalente: O parâmetro {minResourceURI} é omitido e o parâmetro [NVStorageURI] recebe o valor “[all]”. e) Para o comando saveApplication o parâmetro location deve receber o valor “intMem” f) Quando iniciada, a aplicação NCL deve obrigatoriamente ser iniciada a partir de todas as interfaces do elemento <body>.
STORE (0x08)	A aplicação não é iniciada automaticamente. Ela é colocada em uma lista de aplicações disponíveis no receptor, podendo ser selecionada por um usuário para ser então iniciada. A aplicação pode ser armazenada previamente a sua execução.	Semelhante ao AUTOSTART, exceto que a aplicação é iniciada pelo usuário e não automaticamente.	Ver AUTOSTART
STORED_AUTOSTART (0x09)	São similares a aplicativos sinalizados com AUTOSTART, exceto que devem ser completamente	addApplication (baseId, {uri, id}, applicationId, {all}, [all]);	a) A base privada é identificada pelo “valor do campo network_id da NIT”, ou o “valor do

	armazenados de forma persistente no receptor antes da sua execução	seguido de saveApplication (baseId, applicationId, intMem) seguido de startDocument (baseId, documentId, interfaceId, 0, null, null) para adicionar (se já não estiver adicionada) e iniciar uma aplicação na base privada associada com o serviço sintonizado	campo program_number da PMT que refere à AIT"; b) O atributo id do elemento <ncl>, o parâmetro applicationId dos comandos addApplication e saveApplicaion, e o parâmetro documentId do comando startDocument devem receber o mesmo valor do campo application_id da estrutura application_identifier da AIT; c) Para o comando addApplication equivalente: Há apenas um parâmetro uri que deve obrigatoriamente receber o valor definido no campo base_directory do ginga_ application_location_descriptor. Há apenas um parâmetro de id que deve obrigatoriamente receber o valor IOR correspondente ao documento definido no campo initial_entity do ginga_ application_location_descriptor. d) Para o comando addApplication equivalente: O parâmetro {minResourceURI} recebe o valor "{all}", e o parâmetro [NVStorageURI] recebe o valor "[all]". e) Para o comando saveApplication o parâmetro location deve receber o valor "intMem". f) A aplicação NCL deve obrigatoriamente ser

			iniciada a partir de todas as interfaces do elemento <body>; g) A aplicação NCL deve obrigatoriamente ser iniciada imediatamente, já que não pode ter seu tempo inicial se referindo a um valor NPT, que supõe-se ser "do it now"; h) A aplicação NCL deve obrigatoriamente ser iniciada do seu começo, já que o parâmetro *offset* é assumido como "0".
STORED_PRESENT (0x0A)	A aplicação não é iniciada automaticamente. Ela é colocada em uma lista de aplicações disponíveis no receptor, podendo ser selecionada por um usuário para ser então iniciada. A aplicação deve ser completamente armazenada de forma persistente previamente a sua execução.	Semelhante ao STORED_AUTOSTART, exceto que a aplicação é iniciada pelo usuário e não automaticamente.	Ver STORED_AUTOSTART
STORED_REMOVE (0x0B)	Remove a aplicação. A aplicação deve ser finalizada antes de sua remoção, caso esteja em execução.	removeApplication (baseId, applicationId)	a) A base privada é identificada pelo “valor do campo network_id da NIT”, ou o “valor do campo program_number da PMT que refere à AIT”; b) O atributo id do elemento <ncl> e o parâmetro applicationId do comando removeApplication devem receber o mesmo valor do campo application_id da estrutura application_identifier da AIT.

Página 103, Seção 9.1 Tabela 57

Adicionar após a 3ª. linha da Tabela:

applicationId	Atributo *id* do elemento <ncl> do documento NCL que é o ponto de entrada da aplicação.

Página 114, Seção 9.2

Inserir após o término da Seção 9.2:

9.3 Gerenciamento de Aplicações

9.3.1 Recepção de aplicações

Os componentes das aplicações podem ser entregues por diferentes meios. Os seguintes casos podem ser identificados:

1. Aplicações confinadas a um canal de TV. Nesse caso, todos os componentes das aplicações são entregues pela rede de difusão terrestre (canal de *broadcast*).
2. Aplicações remotas. Nesse caso, todos os componentes das aplicações são entregues a partir de um servidor ligado à Internet (canal *broadband*).
3. Aplicações instaladas. Aplicações entregues através de um dispositivo persistente, por exemplo, um *pen drive*.
4. Aplicações residentes. Aplicações embarcadas no receptor pelo seu fabricante.
5. Aplicações híbridas. Aquelas onde parte da aplicação é entregue pela rede de difusão terrestre (canal de *broadcast*) e outra parte entregue através de um dispositivo persistente ou buscada sob demanda na rede de banda larga (Internet).

Dependendo do perfil do receptor componentes das aplicações podem ser ignorados. Por exemplo, os componentes das aplicações dos casos 3 e 4 acima podem ser ignorados por receptores SBTVD seguindo estritamente os perfis A e B.

Como especificado na Seção 9.1, aplicações cuja recepção é sinalizada pela rede de difusão terrestre devem ser colocadas na estrutura de dados PBDS na base privada associada ao canal sintonizado ou em uma base privada aninhada na base privada do canal sintonizado; aplicações residentes são filhas da base específica para aplicações residentes; e aplicações instaladas são filhas da base específica para aplicações instaladas.

Toda estrutura de dados PBDS para gerenciamento das aplicações é mantida em memória volátil ou não volátil. No espaço de armazenamento deve haver sempre um número mínimo de memória livre (conforme ABNT NBR 15604), ou ocupada por informações não persistentes, para que aplicações recém recebidas possam ser armazenadas.

Ao ser recebida uma aplicação, não importa por que meio, ela é armazenada, primeiramente, de forma não persistente, exceto aplicações sinalizadas pela AIT com o código de controle da aplicação com valor UNBOUND, STORED_AUTOSTART ou STORED_PRESENT (nesse caso, a aplicação deve ser armazenada de forma persistente, se e somente se houver espaço suficiente na memória não volátil do sistema). Bases privadas e aplicações se tornam persistentes quando são explicitamente salvas. Se, no momento de seu armazenamento, uma aplicação (exceto aquelas sinalizadas pela AIT com o código de controle da aplicação com valor UNBOUND, STORED_AUTOSTART ou STORED_PRESENT) não tiver disponível espaço livre na estrutura de dados das bases privadas, espaço de memória ocupada deve ser liberado removendo-se aplicações não persistentes utilizando-se a política LRU (*Least Recently Used*) em aplicações que não estão sendo executadas. Aplicações de tamanho maior que o número mínimo de memória livre, ou ocupada por informações não persistentes, podem não ter espaço garantido para sua recepção completa.

9.3.2 Gerenciamento do espaço das aplicações

AppCatUI (Application Catalogue User Interface) é uma funcionalidade do middleware Ginga cuja função é listar as aplicações da estrutura de bases privadas que podem ser disparadas pelo usuário, bem como permitir ao usuário adicionar e remover aplicações, torná-las persistentes e iniciar suas

execuções. Todas essas operações são realizadas sobre a estrutura em árvore que representa a base privada. Por não causar nenhum problema de interoperabilidade, a AppCatUI depende apenas do fabricante, que pode torná-la um ponto diferencial de seu produto. No entanto, alguns procedimentos são obrigatórios e outros fortemente recomendados.

A lista deve obrigatoriamente identificar as aplicações que são persistentes e as que não são persistentes. É recomendado que ela identifique também as aplicações cujos recursos mínimos requeridos por elas (informados por meta dados) já se encontram carregados.

É recomendado que a ordem da lista apresentada pelo AppCatUI mude dinamicamente de acordo com as seguintes regras:

- Primeiramente, as aplicações associadas ao canal sintonizado devem ser apresentadas, com destaque, para que o usuário possa claramente identificá-las. Aplicações associadas são aquelas que são parte do serviço de TV digital sintonizado pelo usuário em um dado instante.
- Segundo, a ordem definida das aplicações associadas é definida pela ordem em que elas são declaradas no serviço (por exemplo, na AIT)
- Aplicações instaladas e residentes devem ser listadas depois que todas as aplicações das bases privadas associadas a canais de TV forem listadas.

Aplicações e bases privadas devem obrigatoriamente poder se tornar persistentes (serem salvas) também por meio da AppCatUI. Nesse caso, para tornar persistentes as aplicações associadas, elas devem obrigatoriamente trazer uma permissão para tal procedimento quando recebidas, conforme definido pelas Normas ABNT NBR 15605. Se não trouxerem a permissão, elas não podem ser tornadas persistentes.

A AppCatUI deve obrigatoriamente também permitir listar aplicações armazenadas em dispositivos de armazenamento externos. Deve obrigatoriamente também ser possível mover ou copiar para tais dispositivos aplicações da estrutura de bases privadas que proveem permissão para tal (tal facilidade é definida pela Normas 15605). Aplicações da estrutura de bases privadas sem a permissão, não podem ser movidas ou copiadas para dispositivos de armazenamento externos. Uma aplicação armazenada em dispositivo externo também pode ser copiada para a estrutura de bases privadas, mais precisamente, para a base privada associada às aplicações instaladas.

A AppCatUI deve obrigatoriamente também permitir mover uma aplicação da base privada associada a um canal de TV para a base privada associada às aplicações instaladas, se assim for permitido pela aplicação, conforme definido pelas Normas ABNT NBR 15605. Se não tiverem a permissão, elas não podem ser movidas.

A transferência de uma aplicação da base privada associada às aplicações instaladas para a base privada associada a um canal de TV deve obrigatoriamente ser possível se essa aplicação for assinada e tiver sua integridade garantida pelo middleware. Esse procedimento deve ser efetuado quando da transferência da aplicação da base privada associada ao canal de TV para a base privada associada às aplicações instaladas ou para um dispositivo de armazenamento externo.

Para atender ao caso de uso de transferência de uma aplicação da base privada associada às aplicações instaladas para a base privada associada a um canal de TV além do caso mencionado no parágrafo anterior, é necessário que a aplicação tenha uma permissão associada, conforme definido pelas Normas 15605.

A AppCatUI deve obrigatoriamente sempre permitir a remoção de aplicações da estrutura de bases privadas pelo usuário, sem restrições.

Note que a liberação do espaço da memória não volátil das aplicações persistentes fica ao encargo do usuário final. Como uma aplicação tem acesso ao espaço livre, ela pode "convidar" o usuário a liberar espaço, mas a decisão cabe ao usuário (ou por uma configuração do receptor que ele escolheu).

Aplicações residentes não podem ser adicionadas, removidas ou movidas através da AppCatUI.

Nos perfis de receptores que não suportam aplicações instaladas ou residentes, e nem a persistência de aplicações, a AppCatUI deve trazer apenas a lista das aplicações sintonizadas.

9.3.3 Segurança e autenticação

O middleware não é obrigado a executar aplicações não assinadas, não autenticadas e que não tenham sido sinalizadas pelo canal de TV, todavia, caso o faça, essas estarão restritas à execução local, sem nenhum acesso a dispositivos externos, à Internet e a PBDS.

Para garantir outros direitos a qualquer recurso pelas aplicações, se torna necessário a obediência às Normas 15605.

Página 114, Seção 10.3.1

Inserir após o item d) da lista:

e) módulo *dir*: oferece uma API para manipular diretórios;

Página 129, Seção 10.3.3.3

Remover o 1º. Parágrafo da **Classe tcp**:

O uso do canal de interatividade é realizado por meio dessa classe de eventos.

Página 129, Seção 10.3.3.3

Adicionar o trecho abaixo após toda a descrição da Classe tcp:

Classe http:

Uma aplicação NCLua faz uma requisição HTTP ou HTTPS postando um evento da forma:

evt = {class='http', type='request', method=*method*, uri=*uri*, [session=*session*], [headers=*headers*], [body=*body*], [timeout=*timeout*]},

em que *method* (string) denota o método da requisição ('get', 'head', 'post', 'put', 'delete', 'connect', 'options' ou 'trace'); *uri* (string) denota a URI da requisição; *session* é um objeto Lua qualquer usado para identificar a requisição; *headers* (tabela) contém os campos do cabeçalho da requisição, em que cada campo é um par chave-valor, ambos strings; *body* (string) contém o corpo da requisição; e *timeout* (número) denota o tempo máximo em segundos que a aplicação pretende aguardar por respostas da requisição. Os campos "session", "headers", "body" e "timeout" são opcionais. Se "header" é nulo, assume-se o cabeçalho vazio. Se "body" é nulo, assume-se o corpo vazio. Se "timeout" é nulo, assume-se que a aplicação aguardará indefinidamente por respostas da requisição. O esquema utilizado na URI, "http://" ou "https://", determina o tipo de requisição (HTTP ou HTTPS).

Similarmente, uma aplicação NCLua recebe as respostas de uma requisição HTTP via eventos da forma:

evt = {class='http', type='response', method=*method*, uri=*uri*, code=*code*, [session=*session*], [headers=*headers*], [body=*body*], [finished=*finished*], [error=*error-message*]},

em que os valores dos campos "method", "uri" e "session" são idênticos aos dos campos homônimos do evento de requisição correspondente; *code* (número inteiro) denota o código de retorno (*status code*) da resposta; *headers* (tabela) contém o cabeçalho da resposta; *body* (string) contém o corpo da resposta; *finished* indica o último evento de resposta; e *error-message* (string) contém uma mensagem de erro. Os campos "session", "headers", "body", "finished" e "error" são opcionais. A presença do campo "finished" indica o último evento de resposta (fim da resposta). A presença do campo "error" indica a ocorrência de um erro de comunicação. Note que é possível ocorrer um erro de protocolo (campo "code" indicando erro) sem que haja erro de comunicação (campo "error" nulo). Os campos "finished" e "error" são mutuamente exclusivos.

Uma aplicação NCLua pode cancelar uma requisição pendente postando um evento da forma:

evt = {class='http', type='cancel', [session=*session*]},

em que o valor do campo "session" identifica a requisição a ser cancelada. Se "session" é nulo ou se não há requisição pendente cujo campo "session" é igual a *session*, então o evento de cancelamento é ignorado.

O únicos filtros dependentes da classe "http" são *type* e *session*, nessa ordem.

Página 118, Seção 10.3.2.3, 8º.parágrafo da **canvas:attrCrop (x, y, w, h: number)**

Substituir:

Apenas a região configurada passa a ser usada em operações de composição.

Por:

A região de *crop* é a região usada quando o canvas é a origem (parâmetro *src*) de uma operação de composição (função canvas.compose).

Página 120, Seção 10.3.2.3, **canvas:attrScale (w, h: number)**

Substituir todo o texto da função **por**:

Argumentos

w	Largura de escalonamento do canvas (em pixels)
h	Altura de escalonamento do canvas (em pixels)

Descrição

Altera os atributos de escalonamento horizontal e vertical do canvas.

Os atributos de escalonamento são usados para escalar o canvas quando este é a origem (parâmetro *src*) de uma operação de composição (função canvas.compose).

Um dos parâmetros pode ser "true", indicando que a proporção do canvas deve ser mantida.

Um dos parâmetros pode ser "false", indicando que o valor atual do atributo deve ser mantido.

O canvas principal não pode ter seu valor alterado pois é controlado pelo formatador NCL.

Página 123, Seção 10.3.2.5, **canvas:compose (x, y: number; src: canvas; [src_x, src_y, src_width, src_height: number])**

Substituir os parágrafos 10 e 11:

Faz sobre o canvas (canvas de destino), em sua posição (x,y), a composição pixel a pixel com src (canvas de origem).

Os outros parâmetros são opcionais e indicam que parte do canvas src compor. Quando ausentes o canvas inteiro é composto.

Por:

Se *src_x*, *src_y*, *src_w* e *src_h* forem dados, compõe pixel-a-pixel a região do canvas de origem (*src*) delimitada por *src_x*, *src_y*, *src_width* e *src_height* sobre o canvas de destino (*canvas*), na posição (*x*,*y*). Caso contrário, compõe pixel-a-pixel a região de *crop* do canvas de origem (ver função canvas.attrCrop, Seção 10.3.2.3) sobre o canvas de destino, na posição (*x*,*y*). Em ambos os casos, a operação de composição deve obrigatoriamente considerar atributos *flip*, *opacity*, *rotation* e *scale* do canvas de origem. Isto é, antes de ser composta sobre o canvas de destino, a região de selecionada do canvas de origem (seja aquela delimitada pelos parâmetros *src_x*, *src_y*, *src_width*, e *src_height* ou aquela delimitada pelo atributo *crop*) deve ser obrigatoriamente invertida, opacificada ou transparentada, rodada e escalada de acordo com os valores correntes dos atributos *flip*, *opacity*, *rotation* e *scale* do canvas de origem.

Página 139, Seção 10.3.4

Adicionar no final da Seção

A subtabela “settings.system.pbds” representa a árvore correspondente à estrutura de bases privadas (PBDS). Cada par chave-valor (*k*,*v*) da tabela “settings.system.pbds” denota um filho da raiz da árvore PBDS. A chave *k* do par contém o identificador do nó—isto é, o *baseId* de um canal ou serviço, ou uma das strings pré-definidas “installed”, para aplicações instaladas, ou “resident”, para aplicações residentes—e o valor *v* do par é uma tabela da forma:

baseId={
    applicationId={{*fileURI*}, ..., {*fileURI*}},
    ...
    applicationId={{*fileURI*}, ..., {*fileURI*}},
}

em que *applicationId* é o identificador de uma aplicação do canal ou serviço *baseId*, e *fileURI* é uma string contendo a URI de um arquivo (código, recurso, ou metadado) da aplicação correspondente. Toda subtabela (nó) da tabela “settings.system.pbds” pode conter uma chave “_attributes” cujo valor associado é uma tabela contendo os atributos do nó em questão—isto é, pares chave-valor definidos pela tabela abaixo.

Nó	Atributo	Tipo	Descrição
baseId	persistent	boolean	“true” se o conteúdo da base está armazenado em memória persistente; “false”, caso contrário
	state	string	“open”, se a base está aberta; “active”, se a base está ativa; ou “closed”, se a base está fechada
applicationId	persistent	boolean	“true” se o conteúdo da aplicação está armazenado em

			memória persistente; “false”, caso contrário
	entryPoint	string	URI do código do ponto de entrada da aplicação
	minResources	tabela	Lista das URIs (strings) dos objetos necessários para disparar a aplicação
	completion	boolean	“true” se todos os conteúdos recebidos sem solicitação estão completos; “false”, caso contrário
{*fileURI*} (tabela contida em *applicationId*)	completion	número	Número no intervalo [0,1] que denota a porcentagem do progresso do *download* do objeto *fileURI*, por exemplo 0.3=30%, 0.5=50%, 1.0=100%, etc.

Página 139, Seção 10.3.5

Adicionar no final da Seção 10.3.5, antes da Seção 10.4

10.3.6 Módulo *dir*

Define funções para manipular diretórios.

dir.list (path:string) -> iterator:function

Argumentos

path caminho do diretório

Valor de retorno

iterator função de iteração do diretório

Descrição

Retorna uma função que cada vez que é chamada retorna uma entrada do diretório (string) ou *nil* caso o diretório não possua entradas ou caso todas as entradas já tenham sido retornadas.

dir.test (path:string, [query:string]) ->result:boolean

Argumentos

path caminho do arquivo ou diretório

query teste a ser realizado

Valor de retorno

result resultado do teste

Descrição

Realiza um dos seguintes testes:

- se *query* é igual a “exists”, retorna *true* se *path* é um caminho acessível;
- se *query* é igual a “dir”, retorna *true* se *path* é um diretório;
- se *query* é igual a “regular”, retorna *true* se *path* é um arquivo comum.

Caso o teste falhe, a função retorna *false*; se *query* é nulo a função assume “exists”.