GY

中华人民共和国广播电视行业标准

GY/T 303.5—2018

智能电视操作系统
第5部分：功能组件接口

Smart TV operating system—
Part 5: Functional Component Interface

2018－07－06 发布　　2018－07－06 实施

国家广播电视总局　　发布

目　　次

前　言

GY/T 303《智能电视操作系统》已经或计划发布以下部分：

——第1部分：功能与架构；

——第2部分：安全；

——第3部分：应用程序编程接口；

——第4部分：硬件抽象接口；

——第5部分：功能组件接口；

——第6部分：可信执行环境接口；

——第7部分：符合性测试。

本部分为GY/T 303的第5部分。

本部分按照GB/T 1.1—2009给出的规则起草。

本部分由全国广播电影电视标准化技术委员会（SAC/TC 239）归口。

本部分起草单位：国家新闻出版广电总局广播科学研究院、国家广播电视网工程技术研究中心、华为技术有限公司、四川长虹网络科技有限责任公司、深圳市海思半导体有限公司、四川九州电子科技股份有限公司、深圳市茁壮网络股份有限公司、东方有线网络有限公司、深圳创维-RGB电子有限公司、北京数码视讯科技股份有限公司、杭州国芯科技股份有限公司、上海高清数字科技产业有限公司、北京永新视博数字电视技术有限公司、上海联彤网络通讯技术有限公司、中兴通讯股份有限公司、北京数字太和科技有限责任公司、湖南国科微电子股份有限公司、国家新闻出版广电总局卫星直播管理中心、江苏银河电子股份有限公司、江苏省广电有线信息网络股份有线公司、中国有线电视网络有限公司。

本部分主要起草人：盛志凡、黎政、同磊、咎元宝、程伯钦、严海峰、蒋艳山、李洪浩、杨明磊、万乾荣、马万铮、袁宏伟、来永胜、王旭升、解伟、郭沛宇、赵良福、王强、王磊、郭晓霞、王明敏、杨勋、白伟、张晶、赵学庆、何剑、郝建伟、董进刚、梁志坚、王继刚、郭永伟、赵鹏、郑力铮、刘锦阳、李小雨、王东飞、王欣刚、王佳敏、李玮帆、贾汇东、张雷鸣、张伟、施玉海、付瑞、张定京、王颖、汤新坤、万倩、贾庭兰、朱里越、林宝成、白鹤、谌颖、杨旭、李爽、刘江。

引　言

本部分的发布机构提请注意，声明符合本部分时，可能使用涉及本部分有关内容的相关授权的和正在申请的专利如下：

序号	章条号	专利名称
1	4、5、6	一种智能电视操作系统
2	4、5、6	一种智能电视系统
3	6.3、附录C	一种在智能电视操作系统中支持全媒体播放的方法及智能电视终端
4	6.6、附录F	一种用于智能操作系统的条件接收方法和系统（201510882112.9）
5	6.6、附录F	一种用于智能操作系统的条件接收方法和系统（201510884736.4）
6	6.5、附录E	一种用于智能操作系统的数字版权管理（DRM）方法和系统
7	6.5、附录E	一种支持数字版权管理（DRM）的媒体网关/终端实现方法及其设备

本部分的发布机构对于该专利的真实性、有效性和范围无任何立场。

该专利持有人已向本部分的发布机构保证，他愿意同任何申请人在合理且无歧视的条款和条件下，就专利授权许可进行谈判。该专利持有人的声明已在本部分的发布机构备案，相关信息可以通过以下联系方式获得：

专利权利人	联系地址	联系人	邮政编码	电话	电子邮箱
国家新闻出版广电总局广播科学研究院	北京市西城区复兴门外大街2号	孟祥昆	100866	010-86098010	mengxiangkun@abs.ac.cn

请注意除上述专利外，本部分的某些内容仍可能涉及专利。本部分的发布机构不承担识别这些专利的责任。

智能电视操作系统
第5部分：功能组件接口

1 范围

GY/T 303的本部分规定了智能电视操作系统的功能组件接口相关技术要求。

本部分适用于智能电视操作系统功能组件接口的研发、生产、应用和测试。

2 规范性引用文件

下列文件对于本部分的应用是必不可少的。凡是注日期的引用文件，仅所注日期的版本适用于本部分。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本部分。

GB/T 28160—2011 数字电视广播电子节目指南规范

GY/T 255—2012 可下载条件接收系统规范

GY/T 303.1—2016 智能电视操作系统 第1部分：功能与架构

GY/T 303.2—2016 智能电视操作系统 第2部分：安全

W3C HTML 5.2 超文本标记语言5.2（Hyper Text Markup Language 5.2）

W3C CSS 2.1 级联样式表2级修订1(CSS 2.1)规范（Cascading Style Sheets Level 2 Revision 1（CSS 2.1）Specification）

W3C DOM 2.1 文档对象模型（DOM2级）HTML规范 第1版（Document Object Model Level 2 HTML Specification Version 1.0）

3 缩略语

下列缩略语适用于本部分。

App 应用程序（Application）

API 应用程序编程接口（Application Programming Interface）

ATV 模拟电视（Analog Television）

BAT 业务群关联表（Bouquet Association Table）

CA 认证机构（Certification Authority）

DCAS 可下载条件接收系统（Downloadable Conditional Access System）

DOM 文档对象模型（Document Object Model）

DRM 数字版权管理（Digital Rights Management）

DTV 数字电视（Digital Television）

DTVAL 数字电视适应层（DTV Adaptation Layer）

DVB 数字视频广播（Digital Video Broadcasting）

ECM 授权控制信息（Entitlement Control Message）

EIT 事件信息表（Event Information Table）

EMM 授权管理信息（Entitlement Management Message）
EPG 电子节目指南（Electronic Program Guide ）
ES 基本码流（Elementary Stream）
HTML 超文本标记语言（Hyper Text Markup Language）
JS JavaScript脚本语言（Java Script）
TVOS-H 基于HTML的TVOS（TV Operating System-HTML）
TVOS-J 基于Java的TVOS（TV Operating System-Java）
NIT 网络信息表（Network Information Table）
NVM 固定存储器（NonVolatile Memory）
NVOD 准互动点播系统（Near Video On Demand）
OSD 屏幕叠加显示（On-Screen Display）
PAT 节目关联表（Program Association Table）
PID 包识别码（Packet Identifier）
PMT 节目映射表（Program Map Table）
PSI 节目特定信息（Program Specific Information）
RAM 随机存取存储器（Ramdom Access Memory）
SDT 业务描述表（Service Descriptor Table）
SI 业务信息（Service Information）
TApp 可信应用（Trusted Application）
TEE 可信执行环境（Trusted Execution Environment）
TS 传送流（Transport Stream）
URL 统一资源定位符（Uniform Resource Locator）

4 功能组件接口概述

TVOS 功能组件应向应用框架层功能接口单元和组件层其他功能组件提供调用接口。

TVOS 功能组件接口应包括组件服务管理、数字电视、媒体引擎、HTML5 引擎、DRM、DCAS、人机交互、多屏互动、广播信息服务、ATV、应用管理、消息管理等功能组件接口。如图 1 所示。

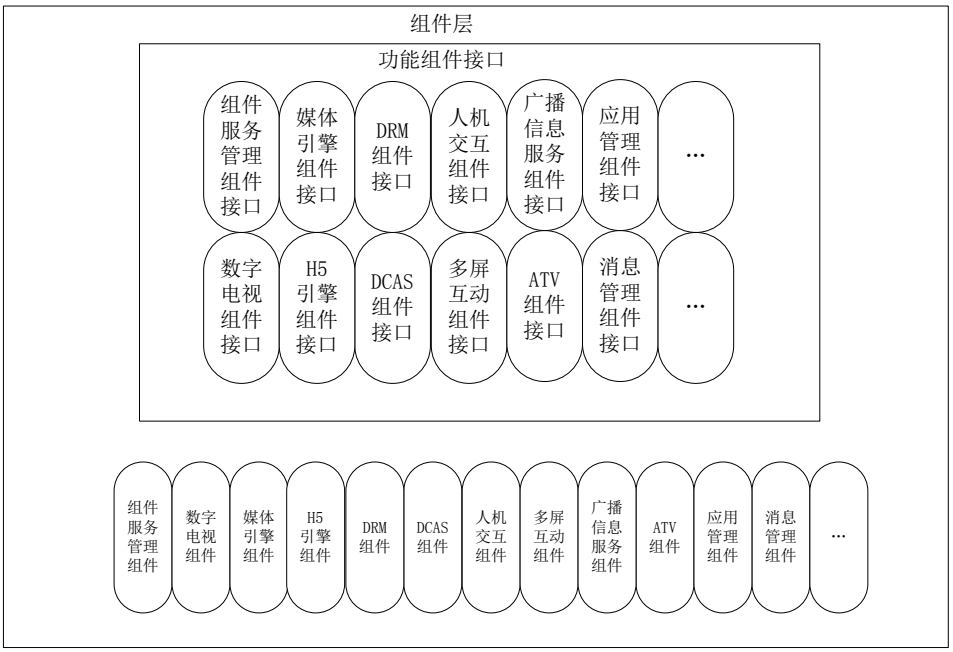

图 1　TVOS 功能组件接口

TVOS 功能组件接口应包含向应用框架层提供的软件调用接口以及向其它功能组件提供的软件调用接口。TVOS 公共功能组件接口应支持被应用框架层封装为 Java 和 Web 两类应用编程接口。TVOS 功能组件变更应保持组件接口的前向兼容。TVOS 功能组件接口简表见表 1。

TVOS 功能组件接口降低功能组件层和应用框架层之间的耦合，既起到架构看护作用，又为系统演进提供空间。

表 1　TVOS 功能组件接口

序号	功能组件接口	描述	备注
1	组件服务管理组件接口	提供组件服务管理相关调用接口。	详见 6.1
2	数字电视组件接口	提供调谐、解调和 Tuner 状态监控等功能接口；提供 DVB 单向广播节目搜索、业务信息获取、存储和查询等功能接口；提供单向广播 EPG 数据的获取和解析、以及频道管理等功能接口。	详见 6.2
3	媒体引擎组件接口	提供对各类媒体音视频的播放和控制等功能接口。	详见 6.3
4	HTML5 引擎组件接口	提供 HTML5 网页的加载和解析等功能接口。	详见 6.4
5	DRM 组件接口	提供 DRM App 注册、注销和运行状态等管理接口；提供 DRM App 与 DRM TApp 之间的消息传递接口；提供媒体引擎组件与 DRM App 和 DRM TApp 之间的消息传递接口。	详见 6.5
6	DCAS 组件接口	提供 DCAS App 注册和管理等功能接口；提供与 DTV 组件和媒体引擎组件协同实现带内传输条件接收授权控制信息和授权管理信息的接收和转发等功能接口；提供与相关网络协议栈模块协同实现带外传输条件接收授权管理信息的接收和转发功能接口；提供 CA 版本、Chip ID 和授权状态等 CA 相关信息的查询接口；为 DCAS App 与 DCAS TApp 提供信息交换通道接口。	详见 6.6

表 1（续）

序号	功能组件接口	描述	备注
7	人机交互组件接口	提供对遥控器、键盘、鼠标、游戏手柄和移动终端等输入设备的信息处理接口；提供语音操控输入的信息处理接口。	详见 6.7
8	多屏互动组件接口	提供手机、平板和电视等设备发现及连接功能接口；提供跨屏 UI 操控功能接口。	详见 6.8
9	广播信息服务组件接口	提供广播信息服务、OSD 更新、广告更新、应急广播监测、信息服务、DCAS 数据监测等相关业务功能接口。	详见 6.9
10	ATV 组件接口	提供 ATV 频道搜索和管理、信号源通道管理和 TV 相关设置参数管理等功能接口。	详见 6.10
11	应用管理组件接口	提供应用安装、卸载、启动、停止和应用信息获取等应用管理功能接口。	详见 6.11
12	消息管理组件接口	提供 DCAS 和 DRM 组件与其他组件之间的消息传送接口。	详见 6.12

5 功能组件模型

组件模型原理如图 2 所示。其中，BnFooService 为服务桩代码（Stub），BpFooService 为服务代理（Proxy）。

TVOS 组件应由服务端和客户端组成，服务端和客户端运行在不同的进程空间，且使用 i-Binder 机制实现跨进程通信。组件服务端负责实现相应组件功能并通过硬件抽象层调用内核层软件模块和底层硬件；组件服务端主要包括服务实现和服务 Stub 等软件模块；组件服务端是一个系统常驻的运行实例，一个组件服务端运行实例服务多个不同的组件客户端运行实例；组件客户端主要包括客户端实现、服务 Proxy 和客户端 API 等软件模块。公共功能组件的服务端和客户端均应采用 C/C++编程语言实现。

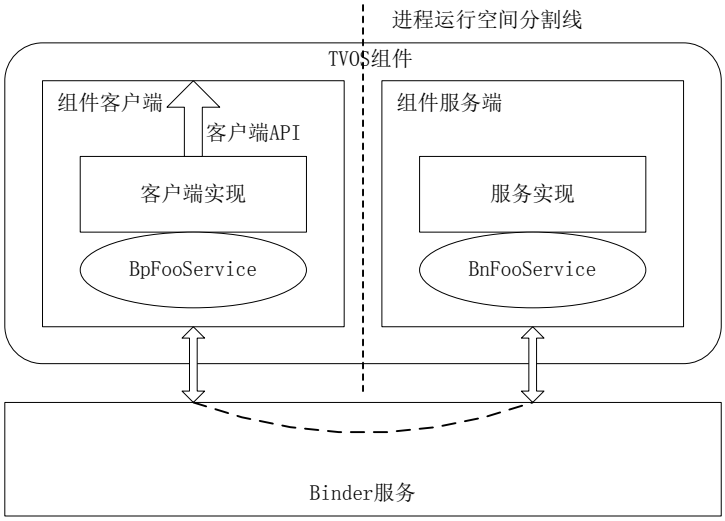

图 2 功能组件模型框图

6 功能组件接口说明

6.1 组件服务管理组件接口

组件服务管理组件遵从 TVOS 功能组件架构，采用 C/S 组件模型，提供组件服务管理功能。组件服务管理服务端在独立的进程中运行，服务进程在系统启动时先于其他组件服务启动，并一直在后台运行。组件服务管理客户端封装成 Client.lib（静态库）和 Client.so（动态库）的形式供其他组件和系统其他部分调用。组件服务管理组件接口的实现应符合 GY/T 303.1—2016 中 8.2 的相关规定。

附录 A 定义了组件服务管理组件对外提供的接口，包括组件服务管理功能模块对外接口。组件服务管理组件接口简表见表 2。

表 2 组件服务管理组件接口

接口	说明
组件服务管理功能模块	
addService	注册一个服务到组件服务管理组件。
checkService	根据服务名称以非堵塞方式获取服务对象。
getService	根据服务名称以堵塞方式获取服务对象。
listServices	获取当前运行中的服务名称列表。

6.2 数字电视组件接口

数字电视组件遵从TVOS功能组件架构，数字电视功能组件的服务端在独立的进程中运行，服务进程在系统启动时启动，并一直在后台运行。数字电视功能组件的客户端封装成DTVClient.lib（静态库）和DTVClient.so（动态库）的形式供其他组件和系统其他部分调用。数字电视组件接口的实现应符合GY/T 303.1—2016中8.3的相关规定。

附录 B 定义了数字电视组件对外提供的接口，包括调谐解调、节目搜索、数据过滤、广播协议信息查询、电子节目指南和节目信息管理等功能模块对外接口。数字电视组件接口简表见表 3。

表 3 数字电视组件接口

接口	说明
调谐解调功能模块	
DTVAL_getAllTunerID	获取所有Tuner的ID。
DTVAL_tuneStream	将Tuner调谐到指定的传送流。
DTVAL_tune	将Tuner调谐到指定的频点。
DTVAL_getSystemDeliveryType	获取Tuner的传送流类型。
DTVAL_getSignalStatus	获取信号状态。
DTVAL_getTunerStatus	获取Tuner的锁定状态。
DTVAL_getCurrentTunningParam	获取当前调谐参数。
DTVAL_getCurrentTransportStream	获取当前流信息。
DTVAL_getCurrentService	获取当前正在播放的业务。
DTVAL_setCurrentService	设置当前正在播放的业务。
节目搜索功能模块	
DTVAL_startScan	开始进行频道搜索。
DTVAL_stopScan	取消频道搜索。

表 3（续）

接口	说明
DTVAL_startScanByJson	下载Json数据并解析。
DTVAL_JSON_Start	解析Json数据，并保存到数据库中。
DTVAL_JSON_Stop	停止Json数据下载解析。
DTVAL_updateScanResult	更新PSI/SI数据。
DTVAL_saveScanResult	保存PSI/SI数据到NVM。
DTVAL_revertScanResult	从NVM导入PSI/SI数据到RAM中。
DTVAL_deleteScanResult	清除RAM和NVM中的PSI/SI数据。
广播协议信息查询与数据过滤功能模块	
DTVAL_getUnusedFilterNumber	获得系统中当前未使用的段过滤器数量。
DTVAL_requestFilter	申请使用一个段过滤器。
DTVAL_releaseFilter	释放占用的段过滤器。
DTVAL_attachStream	将段过滤器与传送流绑定。
DTVAL_detachStream	将段过滤器与其绑定的传送流断开。
DTVAL_startFiltering	开始过滤。
DTVAL_stopFiltering	停止过滤。
DTVAL_startTableMonitor	启动表格更新的监控。
DTVAL_stopTableMonitor	停止对表格更新事件的监控。
DTVAL_getSIInfo	获取SI信息。
DTVAL_releaseSIInfo	释放SI信息。
DTVAL_retrieveActualNetwork	从现行NIT表中获取当前网络信息。
DTVAL_retrieveActualTransportStreams	从现行NIT表中获取当前网络的所有传送流信息。
DTVAL_releaseDVBService	释放DVBService_t*结构体。
DTVAL_releaseDVBTS	释放DVBTS_t* 结构体。
DTVAL_retrieveActualServices	从当前传输流中获取所有业务信息。
DTVAL_retrievePMTService	获取某个业务相关的PMT描述的业务信息。
DTVAL_retrieveTimeFromTDT	从当前传送流承载的TDT表中获取时间信息。
DTVAL_retrieveTimeFromTOT	从当前传送流承载的TOT表中获取时间信息。
电子节目指南功能模块	
DTVAL_EPGManager_getPresentProgram	获取指定业务的当前节目。
DTVAL_EPGManager_getPresentProgramsByContentType	根据参数中指定的节目内容分类值，在当前EPG数据库中查找符合条件的当前节目信息。
DTVAL_EPGManager_getPresentProgramsByName	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的当前节目信息。
DTVAL_EPGManager_getFollowingProgram	获取后续节目信息。
DTVAL_EPGManager_getFollowingProgramsByContentType	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的后续节目信息。
DTVAL_EPGManager_getFollowingProgramsByName	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的后续节目信息。

表3（续）

接口	说明
DTVAL_EPGManager_getProgramsByService	根据参数中指定的业务信息,获取指定业务中符合条件的节目信息。
DTVAL_EPGManager_getProgramsByDate	根据参数中指定的起始日期和结束日期,获取指定业务中符合条件的节目信息。
DTVAL_EPGManager_getProgramsByDirection	根据参数中指定的起始日期和检索方向,获取指定业务中指定个数的节目信息。
DTVAL_EPGManager_getProgramsByContentType	根据参数中指定的节目内容分类值，在当前EPG数据库中查找符合条件的节目信息。
DTVAL_EPGManager_getProgramsByName	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的节目信息。
DTVAL_EPGManager_getReferenceEvents	获取到搜索的ReferenceEvent对象数组。
DTVAL_EPGManager_getReferencePrograms	获取指定参考业务上的参考节目。
DTVAL_ReferenceEvent_getSchedules	获取该参考事件从此时此刻起,若干天之内的所有时移事件列表，当天已经播放完毕的时移事件将被丢弃，时移事件按照起始时间进行排序。
DTVAL_ReferenceEvent_getPresentSchedules	获取该参考事件所带的当前正播放的所有时移事件对象，数组中的元素按照播放起始时间进行排序。
DTVAL_ReferenceEvent_getFollowingSchedules	获取所有的时移事件信息。
DTVAL_EPGManager_getAllProgramServices	返回所有的Service节目信息。
DTVAL_releaseJS_PROGRAMEVENT_PRIVATE	释放JS_PROGRAMEVENT_PRIVATE结构体。
节目信息管理功能模块	
DTVAL_ChannelManager_getChannelByChannelID	根据频道ID获取频道对象。
DTVAL_ChannelManager_getChannelByLogicalID	根据逻辑频道号获取频道对象。
DTVAL_ChannelManager_getChannelByServiceID	根据业务ID获取频道对象。
DTVAL_ChannelManager_getLastChannel	获取前一个打开的指定业务类型的频道。
DTVAL_ChannelManager_getShutDownChannel	获取指定类型的关机频道。
DTVAL_ChannelManager_delChannel	从频道列表中删除指定的频道。
DTVAL_ChannelManager_deleteAllChannels	从频道列表中删除所有的频道。
DTVAL_ChannelManager_deleteAllFavorites	从频道列表中删除所有的喜爱频道。
DTVAL_ChannelManager_deleteAllDelMarkedChannels	从频道列表中删除所有带删除标记的频道。
DTVAL_ChannelManager_resetProperties	将用户所有设置为喜爱、锁定、隐藏等标记的频道全部重置,所有的频道全部更改为非喜爱、非锁定、非隐藏。
DTVAL_ChannelManager_swapChannel	交换频道对象obj1和频道对象obj2在频道列表中的位置。
DTVAL_ChannelManager_sortChannels	按照指定的方式进行频道排序。
DTVAL_ChannelManager_filterChannels	在当前频道列表中过滤出指定条件的新的频道列表。
DTVAL_ChannelManager_saveChannels	将RAM中的频道列表数据保存到NVM中。
DTVAL_ChannelManager_restoreChannels	将NVM中的频道列表数据恢复到RAM中。

表 3（续）

接口	说明
DTVAL_ChannelManager_getServiceByChannel	获取当前频道对象对应的DvbService对象。
DTVAL_ChannelManager_updateChannel	将应用设置的channel属性同步到数据库中。
DTVAL_releaseJS_CHANNEL_PRIVATE	释放JS_CHANNEL_PRIVATE结构体。
DTVAL_ChannelManager_getTunerIDOfLastChannel	获取最后保存节目的TunerID。
DTVAL_SetCarriesList	存储频点列表。
DTVAL_SetServiceList	存储节目列表。
DTVAL_UpdateChannelData	更新节目信息。
DTVAL_ChannelManager_getChannelByServiceType	根据业务类型获取频道对象。

6.3 媒体引擎组件接口

媒体引擎组件遵从TVOS功能组件架构，采用C/S组件模型，媒体引擎组件的服务端在独立的进程中运行，服务进程在系统启动时启动，并一直在后台运行。媒体引擎组件的客户端封装成libmedia.so（动态库）的形式供其他组件和系统其他部分调用。媒体引擎组件接口的实现应符合GY/T 303.1—2016中8.4的相关规定。

附录 C 定义了媒体引擎组件对外提供的接口，包括媒体播放功能模块对外接口。媒体引擎组件接口简表见表 4。

表 4 媒体引擎组件接口

接口	说明
媒体播放功能模块	
setDataSource	以 path 和 Hash Key 对的方式设置数据源。
setDataSource	以文件句柄方式设置数据源。
setVideoSurfaceTexture	设置视频显示画幕。
setListener	设置媒体播放监听器。
prepare	同步的方式让播放器准备，直到播放器准备好才返回。
prepareAsync	异步的方式让播放器准备，调用后该接口立刻返回。
start	从 pause 或者 stop 状态启动播放。
stop	停止播放。
pause	暂停播放。
isPlaying	获取播放器状态是否处于播放中。
getVideoWidth	获取视频播放窗口宽度。
getVideoHeight	获取视频播放窗口高度。
setVideoArea	设置视频播放窗口区域。
getVideoArea	获取视频播放窗口区域。
seekTo	让播放器在指定的位置播放。
getCurrentPosition	获取当前播放位置。
getDuration	获取播放源的可播放长度。
reset	重置播放器。

表 4（续）

setPace	设置播放器速率。
getPace	获取播放器速率。
setStopMode	设置媒体播放暂停效果。
getStopMode	获取媒体播放暂停效果。
setClip	设置视频源图像的剪切区域。
getClip	获取视频源图像的剪切区域。
getStartTime	获取时移（或者回看）节目的起始时间。
selectAudioStream	选择当前播放器的音频流。
setLooping	设置播放器是否循环播放。
isLooping	获取当前状态是否为循环播放。
setDisplayMode	设置播放器显示模式。
setDisplayRatio	设置播放时视频的宽高比率。
setMetadataFilter	设置媒体元数据过滤器。
getMetadata	获取媒体元数据。
setVideoDisplay	设置视频输出。
getVideoDisplay	获取视频输出状态。

6.4 HTML5 引擎组件接口

HTML5引擎组件遵从TVOS功能组件架构，采用C/S组件模型，提供HTML5网页支撑功能。HTML5引擎功能组件的服务端在独立的进程中运行，服务进程由客户端启动与销毁。HTML5引擎功能组件的客户端封装成HTML5Client.so（动态库）的形式供其他组件和系统其他部分调用。HTML5引擎组件接口的实现应符合GY/T 303.1—2016中8.5的相关规定。

附录 D 定义了 HTML5 引擎组件对外提供的接口，包括 HTML5 功能模块对外接口。HTML5 引擎组件接口简表见表 5。

表 5 HTML5 引擎组件接口

接口	说明
HTML5 功能模块	
browser_main	启动浏览器服务。
page_open	打开一个新的页面。
page_load	加载网址。
page_reload	刷新当前页面。
page_stop	停止加载当前页面。
page_backForward	对当前页面进行前进后退操作。
page_close	关闭当前页面。
page_setPageCreateCallback	设置页面创建完成回调函数。
page_setStartLoadCallback	设置页面开始加载回调函数。
page_setFinishLoadCallback	设置页面结束加载回调函数。
page_setFailLoadCallback	设置页面加载失败回调函数。

6.5 DRM 组件接口

DRM组件遵从TVOS功能组件架构，采用C/S组件模型。DRM组件应实现对DRM App注册、注销和运行状态的管理，实现DRM App与DRM TApp之间的消息传递，实现媒体引擎组件与DRM App和DRM TApp之间的消息传递，支持DRM App和DRM TApp与媒体组件协同实现对加密媒体流的解密，为应用框架层功能接口单元和功能组件层的其他组件提供DRM调用接口。DRM组件接口的实现应符合GY/T 303.1—2016中8.6和GY/T 303.2—2016的相关规定。

附录 E 定义了 DRM 组件对外提供的接口，包括 DRM 功能模块对外接口。DRM 组件接口简表见表 6。

表 6 DRM 组件接口

接口	说明
DRM 功能模块	
CHDRMAPI_RegisterApp	注册。
CHDRMAPI_RegisterApp	注册拓展，可以自定义私有的与 TApp 通信的查询许可证和解密命令 ID。
CHDRMAPI_UnRegisterApp	注销。
CHDRMAPI_SendCommandToTEE	发送命令到 TEE。
CHDRMAPI_SendMessageToPlayer	通知播放器消息。
CHDRMAL_CreateCryptoSession	创建解密句柄。
CHDRMAL_DestroyCryptoSession	销毁解密句柄。
CHDRMAL_UpdateCryptoSession	更新 DRM 信息。
CHDRMAL_CheckRights	查询许可证信息。
CHDRMAL_Decrypt	内容解密。
CHDRMAPI_SetLicenseReq_Callback	注册许可证监听函数。
CHDRMAPI_SetDecryptReq_Callback	注册解密监听函数。
CHDRMAPI_SetMessage_Callback	注册消息监听函数。
CHDRMAL_SetPlayerEventListenser	注册播放器时间监听函数。

6.6 DCAS 组件接口

DCAS 组件与 DTV 组件协同实现带内传输条件接收授权控制信息和授权管理信息的接收和转发，与相关网络协议栈模块协同实现带外传输条件接收授权管理信息的接收和转发，为 DCAS App 与 DCAS TApp 提供信息交换通道，支撑媒体引擎组件与 DCAS App 和 DCAS TApp 实现信息交互，实现对 DCAS App 的注册和管理，支持 CA 版本、Chip ID 和授权状态等 CA 相关信息的查询。DCAS 组件接口的实现应符合 GY/T 303.1—2016 中 8.7、GY/T 303.2—2016 和 GY/T 255—2012 的相关规定。

附录 F 定义了 DCAS 组件对外提供的接口，包括 CA 应用功能模块和解扰操作功能模块对外接口。DCAS 组件接口简表见表 7。

表 7 DCAS 组件接口

接口	说明
CA 应用功能模块	
DCASAL_registerCASModule	注册 DCAS 回调函数。
DCASAL_removeCASModule	删除 DCAS 模块的注册。

表 7（续）

接口	说明
DCASAL_enableDescramblingRequests	启动接收解扰请求。
DCASAL_disableDescramblingRequests	停止接收解扰请求。
DCASAL_startEcmLoading	启动接收 ECM。
DCASAL_stopEcmLoading	停止接收 ECM。
DCASAL_startInbandEmmLoading	启动接收带内 EMM。
DCASAL_stopInbandEmmLoading	停止接收带内 EMM。
DCASAL_sendCommandToTEE	发送命令到 TEE 中。
DCASAL_getChipPublicId	获取 ChipId。
DCASAL_sendDescramblingEvent	发送解扰状态。
DCASAL_setData	存储数据接口。
DCASAL_getDSD	获得 DSD 句柄。
DCASAL_registerDSDListener	注册 DSDListener。
DCASAL_removeDSDListener	删除 DSDListener。
DCASAL_getData	读取数据接口。
DCASAL_setCAInfo	设置接口。
DCASAL_getCAInfo	查询接口。
解扰操作功能模块	
DCASAL_startDescrambling	开始解扰接口。
DCASAL_updateDescrambling	更新解扰接口。
DCASAL_stopDescrambling	停止解扰接口。
DCASAL_notifyFreqChange	频点切换通知。

6.7 人机交互组件接口

人机交互组件应实现对遥控器、键盘、鼠标、游戏手柄和移动终端等输入设备的输入信息处理；对语音操控输入和传感器输入的信息处理；对基于游戏的触屏输入与游戏手柄输入的适配；对上层软件模块和其他组件提供调用接口。人机交互组件接口的实现应符合GY/T 303.1—2016中第8章和第10章的相关规定。

附录 G 定义了人机交互组件对外提供的接口，包括键盘与鼠标消息处理和语音消息处理功能模块对外接口。人机交互组件接口简表见表 8。

表 8 人机交互组件接口

接口	说明
键盘与鼠标消息处理功能模块	
getDeviceClasses	获取设备类别。
getDeviceIdentifier	获取设备描述信息。
getDeviceControllerNumber	获取设备对应的控制号。
getConfiguration	获取设备对应的配置信息。
getAbsoluteAxisInfo	获取绝对轴信息。

表8（续）

接口	说明
hasRelativeAxis	判断是否有相对轴。
hasInputProperty	判断设备是否具有某属性。
mapKey	将系统层的按键映射到应用层的定义。
mapAxis	将系统层的原始轴信息映射到应用层的定义。
setExcludedDevices	设置需要忽略的设备。
getEvents	获取设备原始事件。
getScanCodeState	获取指定设备的指定扫描码的状态。
getKeyCodeState	获取指定设备的指定按键的状态。
getSwitchState	获取指定开关的状态。
getAbsoluteAxisValue	获取指定设备的指定轴的绝对值。
markSupportedKeyCodes	标记支持的按键。
hasScanCode	获取指定设备是否支持指定扫描码。
hasLed	获取指定设备是否有指定LED。
setLedState	设置指定设备的指定LED的状态。
setFrontPanelString	设置前面板的显示字符。
getVirtualKeyDefinitions	获取指定设备的虚拟按键定义。
getKeyCharacterMap	获取指定设备的按键映射表。
setKeyboardLayoutOverlay	设定指定设备的按键映射表。
vibrate	设置指定设备的震动时间。
cancelVibrate	取消指定设备的震动。
requestReopenDevices	请求重新打开设备。
wake	设备输入唤醒。
语音消息处理功能模块	
hci_vpInstance_create	创建识别实例。
hci_vpRecognize_start	启动语音识别输入。
hci_vpRecognize_stop	停止语音识别输入。
hci_vpRecognize_cancel	取消语音识别输入。
hci_vpRecognize_destroy	销毁语音识别引擎实例。

6.8 多屏互动组件接口

多屏互动组件应实现手机、平板、电视等多设备间图片和视音频多媒体内容的传送和播控操作，实现跨屏UI操控，应支持UPNP、DLNA和Miracast等协议。多屏互动组件接口的实现应符合GY/T 303.1—2016中8.11的相关规定。

附录H定义了多屏互动组件对外提供的接口，包括设备发现及连接功能模块和跨屏UI操控功能模块对外接口。多屏互动组件接口简表见表9。

表 9　多屏互动组件接口

接口	说明
设备发现及连接功能模块	
startMultiScreenClient	启动发起端。
stopMultiScreenClient	与接收端断开连接。
findSPs	发现多屏互动组件服务。
connect	初始化多屏互动组件服务，连接接收端。
setCallBack	设置组件远程接口的回调。
跨屏 UI 操控功能模块	
queryInfo	向接收端发送的请求指令。
execCmd	向接收端发送的执行指令请求。
inputKeyCode	向接收端发送的按键事件指令请求。
notifyAllTarget	向已连接的组件设备发送通知。

6.9　广播信息服务组件接口

广播信息服务组件是TVOS中广播信息服务业务的核心组件，主要包括业务监测和处理、业务数据下载等功能模块，用于支撑应急广播、信息服务、广告图片、文本更新等相关业务。

广播信息服务组件遵从TVOS组件架构，采用C/S组件模型，服务端在独立的进程中运行，服务进程在系统启动时启动，并一直在后台运行。

广播信息服务组件接口的实现应符合GY/T 303.1—2016中8.14的相关规定。

附录 I 定义了广播信息服务组件对外提供的接口，包括广播信息服务业务监测、OSD 更新、广告更新、应急广播监测和信息服务等功能模块的对外接口。广播信息服务组件接口简表见表 10。

表 10　广播信息服务组件接口

接口	说明
广播信息服务业务监测功能模块	
startServer	启动广播信息服务组件监控。
stopServer	停止广播信息服务业务的监控。
setTunerIdToDth	设置高频头标识到广播信息服务组件。
setListener	注册监听广播信息服务组件消息。
restoreFactorySettings	恢复出厂设置。
SaveNITServiceUpdateVersion	对业务更新描述符的版本号进行保存。
DCASM_sendDataToDTH	DCAS 组件将数据传输给广播信息服务组件。
OSD 更新功能模块	
getOsdXmlFile	获取 osd xml 文件下载路径。
getOSDTAGLength	获取相应 tag 对应的文本长度。
getOSDTAGValue	获取相应 tag 对应的文本内容。
广告更新功能模块	
getAdFinishStatus	开机时获取开机广告处理状态。
getAdInfoCnt	根据广告位获取广告数量。

表 10（续）

接口	说明
getAdInfo	根据广告位获取广告内容。
startAdPCT	开始接收实时广告。
应急广播监测功能模块	
stopEMBDAction	停止本次应急广播上报。
信息服务功能模块	
startDataBD	启动信息服务接收。
getDataBDFinishPercent	获取信息服务下载完成的百分比。
stopDataBD	终止信息服务接收。
deleteDataBDFiles	删除下载的信息服务所有文件。

6.10 ATV 组件接口

ATV 功能组件遵从 TVOS 功能组件架构，采用 C/S 组件模型。ATV 功能组件的服务端在独立的进程中运行，服务进程在系统启动时启动，并一直在后台运行。ATV 功能组件的客户端封装成 Client.lib（静态库）和 Client.so（动态库）的形式供其他组件和系统其他部分调用。ATV 组件接口的实现应符合 GY/T 303.1—2016 中 8.15 的相关规定。

附录 J 定义了 ATV 组件对外提供的接口，包括频道管理、通道管理和电视设置等功能模块对外接口。ATV 组件接口简表见表 11。

表 11 ATV 组件接口

接口	说明
频道管理功能模块	
hsTVsetChannel	切换频道。
hsTVautoScan	自动搜台功能。
hsTVmanualScan	手动搜台功能。
hsTVstopScan	取消搜台功能。
hsTVsetAudioSystem	设置声音制式。
hsTVgetAudioSystem	获取声音制式。
hsTVsetColorSystem	设置彩色制式。
hsTVgetColorSystem	获取彩色制式。
hsTVgetProgramInfo	获取频道信息。
通道管理功能模块	
hsTVsetInputSource	切换信号源。
hsTVgetCurInputSource	获取当前信号源。
hsTVgetCurInputSourceStatus	获取当前通道的信号状态。
电视设置功能模块	
SetPictureMode	设置图像模式。
GetPictureMode	获取图像模式。
SetBrightness	设置图像亮度。

表 11（续）

接口	说明
GetBrightness	获取图像亮度。
SetContrast	设置图像对比度。
GetContrast	获取图像对比度。
SetSaturation	设置图像饱和度。
GetSaturation	获取图像饱和度。
SetSharpness	设置图像清晰度。
GetSharpness	获取图像清晰度。
SetHue	设置图像色调。
GetHue	获取图像色调。
setColorTemp	设置色温。
getColorTemp	获取色温。
hsTVsetBacklightCustom	设置背光自定义值。
hsTVgetBacklightCustom	获取背光自定义值。
hsTVsetMute	设置静音。
hsTVgetMute	获取当前静音状态。
hsTVsetVolume	设置音量。
hsTVgetVolume	获取当前音量值。
hsTVsetBalance	设置平衡。
hsTVgetBalance	获取当前平衡值。
hsTVsetSoundMode	设置声音模式。
hsTVgetSoundMode	获取当前声音模式。
hsTVsetSpdifMode	设置 SPDIF 输出模式。
hsTVgetSpdifMode	获取当前 SPDIF 输出模式。
hsTVsetAudioEQGainData	设置均衡。
hsTVgetAudioEQGainData	获取当前均衡设置。
hsTVsetAudioSyncDelay	设置音频同步延时。
hsTVgetAudioSyncDelay	获取音频同步延时。
hsTVsetAutoVolumeControl	设置自动音量控制开关。
hsTVgetAutoVolumeControl	获取自动音量控制开关。

6.11 应用管理组件接口

应用管理组件遵从TVOS功能组件架构，应用管理功能组件的服务端在独立的进程中运行，服务进程在系统启动时启动，并一直在后台运行。应用管理功能组件的客户端封装成AppmanClient.lib（静态库）和AppmanClient.so（动态库）的形式供其他组件和系统其他部分调用。应用管理组件接口的实现应符合GY/T 303.1-2016中8.17的相关规定。

附录 K 定义了应用管理组件对外提供的接口，包括应用管理功能模块对外接口。应用管理组件接口简表见表 12。

表 12　应用管理组件接口

接口	说明
应用管理功能模块	
registerApp	注册应用进程,告知应用管理服务。
requestApp	请求改变应用状态。
Install	安装应用。
uninstall	卸载应用。
removeUserData	删除应用的缓冲。
start	启动应用。
stop	停止应用。
showHome	显示主页屏幕界面。
registerService	注册服务。
setInstallBwList	安装黑白名单。
delInstallBwList	删除黑白名单。
getInstallBwList	获取已经安装的黑白名单。
getPackageInfo	获取安装包信息。
getPackages	获取已安装的应用安装包列表。
getStatus	获取所有应用状态。
getAppStatus	获取指定应用状态。

6.12　消息管理组件接口

消息管理组件用于DCAS组件和其他组件（如TVOS-J/TVOS-H等）之间的消息传送。

附录 L 定义了消息管理组件对外提供的接口，包括消息管理功能模块对外接口。消息管理组件接口简表见表 13。

表 13　消息管理组件接口

接口	说明
消息管理功能模块	
EMAL_AddListener	注册事件的监听器。
EMAL_Notify	发送事件消息。

附 录 A
（规范性附录）
组件服务管理组件

A.1 概述

本附录定义了组件服务管理组件对外提供的接口，包括组件服务管理功能模块对外接口，见表 A.1。

表 A.1 组件服务管理组件功能模块

模块	说明
组件服务管理功能模块	定义了注册、查询和获取服务等功能接口。

A.2 组件服务管理功能模块

A.2.1 概述

本模块定义了注册、查询和获取服务等功能接口，见表A.2。

表 A.2 组件服务管理功能模块接口

接口	说明
addService	注册一个服务到组件服务管理组件。
checkService	根据服务名称以非堵塞方式获取服务对象。
getService	根据服务名称以堵塞方式获取服务对象。
listServices	获取当前运行中的服务名称列表。

A.2.2 常量定义

A.2.2.1 错误码

原型：enum {
    SVC_FAIL = -1,
    SVC_OK = 0,
    SVC_INVALID_PARAM,
    SVC_PERMISSION_DENIED
};
描述：接口返回错误码。
成员：
SVC_FAIL：内部错误；
SVC_OK：执行成功；
SVC_INVALID_PARAM：无效参数；
SVC_PERMISSION_DENIED：权限不足。

A.2.3 事件类型

无。

A.2.4 数据结构

无。

A.2.5 回调函数定义

无。

A.2.6 接口定义

A.2.6.1 addService接口

原型：status_t addService(const String16& name,const sp<IBinder>& service,
bool allowIsolated = false);
功能：注册一个服务到组件服务管理组件。
参数：name--输入参数，注册的服务名称；
service--输入参数，注册的服务对象；
allowIsolated--输入参数，是否允许独立沙盒进程的访问。
返回：status_t，正常则返回0，否则返回错误码。

A.2.6.2 checkService接口

原型：sp<IBinder> checkService(const String16& name) const;
功能：尝试从组件服务管理组件获取指定的服务对象，如果获取失败立即返回。
参数：name--输入参数，需要获取的服务名称。
返回：sp<IBinder>，正常则返回获取到的服务对象，否则返回NULL。

A.2.6.3 getService接口

原型：sp<IBinder> getService(const String16& name) const;
功能：从组件服务管理组件获取指定的服务对象，会阻塞直到获取成功或超时。
参数：name--输入参数，需要获取的服务名称。
返回：sp<IBinder>，正常则返回获取到的服务对象，否则返回NULL。

A.2.6.4 listServices接口

原型：Vector<String16> listServices();
功能：从组件服务管理组件获取当前运行中的服务列表。
参数：无。
返回：Vector<String16>，返回的服务列表数组。

附 录 B
（规范性附录）
数字电视组件

B.1 概述

本附录定义了数字电视组件对外提供的接口，包括调谐解调、节目搜索、数据过滤、广播协议信息查询、电子节目指南和节目信息管理等功能模块对外接口，见表 B.1。

表 B.1 数字电视组件功能模块

模块	说明
调谐解调功能模块	定义了调谐解调控制相关的功能接口。
节目搜索功能模块	定义了单向广播节目信息搜索功能接口。
广播协议信息查询与数据过滤功能模块	定义了从传输流中查询相关信息和过滤通过DVB标准传输的各种数据的功能接口。
电子节目指南功能模块	定义了电子节目指南信息查询相关功能接口。
节目信息管理功能模块	定义了节目信息查询和管理相关功能接口。

B.2 调谐解调功能模块

B.2.1 概述

本模块定义了调谐解调控制相关的功能接口，见表B.2。

表 B.2 调谐解调功能模块接口

接口	说明
DTVAL_getAllTunerID	获取所有Tuner的ID。
DTVAL_tuneStream	将Tuner调谐到指定的传送流。
DTVAL_tune	将Tuner调谐到指定的频点。
DTVAL_getSystemDeliveryType	获取Tuner的传送流类型。
DTVAL_getSignalStatus	获取信号状态。
DTVAL_getTunerStatus	获取Tuner的锁定状态。
DTVAL_getCurrentTunningParam	获取当前调谐参数。
DTVAL_getCurrentTransportStream	获取当前流信息。
DTVAL_getCurrentService	获取当前正在播放的业务。
DTVAL_setCurrentService	设置当前正在播放的业务。

B.2.2 常量定义

B.2.2.1 错误码

```
原型：enum {
    DTVAL_FAIL1 = -1,
```

```
    DTVAL_OK = 0,
    DTVAL_INVALID_PARAM,
    DTVAL_FAIL2,
    DTVAL_TIMEOUT
};
```

描述：DTVAL层接口返回值。

成员：

DTVAL_FAIL1：操作失败1；

DTVAL_OK ：操作成功；

DTVAL_INVALID_PARAM：无效参数；

DTVAL_FAIL2：操作失败2；

DTVAL_TIMEOUT：操作超时。

B.2.2.2 单向广播网络接入类型

```
原型：typedef enum {
    SATELLITE_DELIVERY_SYSTEM = 0x0,
    CABLE_DELIVERY_SYSTEM = 0x1,
    TERRESTRIAL_DELIVERY_SYSTEM = 0x2,
    ABSS_DELIVERY_SYSTEM = 0xA,
    DTMB_DELIVERY_SYSTEM = 0xC,
} DVB_DELIVERY_TYPE_e;
```

描述：传输类型枚举值。

成员：

SATELLITE_DELIVERY_SYSTEM：卫星传输类型；

CABLE_DELIVERY_SYSTEM：有线传输类型；

TERRESTRIAL_DELIVERY_SYSTEM：地面数字电视传输类型；

ABSS_DELIVERY_SYSTEM：直播卫星传输类型；

DTMB_DELIVERY_SYSTEM：DTMB传输类型。

B.2.2.3 调制方式

```
原型：typedef enum {
    DVB_MOD_TYPE_DEFAULT=0,
    DVB_MOD_TYPE_QAM_16,
    DVB_MOD_TYPE_QAM_32,
    DVB_MOD_TYPE_QAM_64,
    DVB_MOD_TYPE_QAM_128,
    DVB_MOD_TYPE_QAM_256,
    DVB_MOD_TYPE_QAM_512,
    DVB_MOD_TYPE_BPSK,
    DVB_MOD_TYPE_QPSK,
    DVB_MOD_TYPE_DQPSK,
    DVB_MOD_TYPE_8PSK,
```

```
    DVB_MOD_TYPE_16APSK,
    DVB_MOD_TYPE_32APSK,
    DVB_MOD_TYPE_8VSB,
    DVB_MOD_TYPE_16VSB,
    DVB_MOD_TYPE_S_AUTO,
    DVB_MOD_TYPE_BUTT
} DVB_MOD_TYPE_E;
```

描述：DVB调制类型。

成员：

DVB_MOD_TYPE_DEFAULT：默认值；

DVB_MOD_TYPE_QAM_16：QAM16；

DVB_MOD_TYPE_QAM_32：QAM32；

DVB_MOD_TYPE_QAM_64：QAM64；

DVB_MOD_TYPE_QAM_128：QAM128；

DVB_MOD_TYPE_QAM_256：QAM256；

DVB_MOD_TYPE_QAM_512：QAM512；

DVB_MOD_TYPE_BPSK：BPSK；

DVB_MOD_TYPE_QPSK：QPSK；

DVB_MOD_TYPE_DQPSK：DQPSK；

DVB_MOD_TYPE_8PSK：8PSK；

DVB_MOD_TYPE_16APSK：16APSK；

DVB_MOD_TYPE_32APSK：32APSK；

DVB_MOD_TYPE_8VSB：8VSB；

DVB_MOD_TYPE_16VSB：16VSB；

DVB_MOD_TYPE_S_AUTO：S_AUTO；

DVB_MOD_TYPE_BUTT：BUTT。

B.2.2.4 DVB-T TS流优先级

```
原型：typedef enum {
    TUNER_TS_PRIORITY_NONE = 0,
    TUNER_TS_PRIORITY_HP,
    TUNER_TS_PRIORITY_LP,
    TUNER_TS_PRIORITY_INVALID
} TUNER_TS_PRIORITY_E;
```

描述：TS流极性。

成员：

TUNER_TS_PRIORITY_NONE：无优先级；

TUNER_TS_PRIORITY_HP：高优先级；

TUNER_TS_PRIORITY_LP：低优先级；

TUNER_TS_PRIORITY_INVALID：无效值。

B.2.2.5 卫星信号极化方向

原型：typedef enum {
    TUNER_FE_POLARIZATION_H,
    TUNER_FE_POLARIZATION_V,
    TUNER_FE_POLARIZATION_L,
    TUNER_FE_POLARIZATION_R,
    TUNER_FE_POLARIZATION_INVALID,
} TUNER_FE_POLARIZATION_E;
描述：卫星信号极化方向。
成员：
TUNER_FE_POLARIZATION_H：水平极化；
TUNER_FE_POLARIZATION_V：垂直极化；
TUNER_FE_POLARIZATION_L：左旋圆极化；
TUNER_FE_POLARIZATION_R：右旋圆极化；
TUNER_FE_POLARIZATION_INVALID：无效值。

B.2.2.6 信号锁定状态

原型：typedef enum {
    DVB_TUNER_LOCK = 0x0,
    DVB_TUNER_UNLOCK = 0x1,
    DVB_TUNER_TUNING = 0x2
} DVB_TUNER_STATUS_e;
描述：信号锁定状态。
成员：
DVB_TUNER_LOCK:信号锁定；
DVB_TUNER_UNLOCK:信号丢失；
DVB_TUNER_TUNING:信号调制中。

B.2.2.7 TER模式

原型：typedef enum {
    TUNER_TER_MODE_BASE = 0,
    TUNER_TER_MODE_LITE,
    TUNER_TER_MODE_INVALID
} TUNER_TER_MODE_E;
描述：通道中的信号模式。
成员：
TUNER_TER_MODE_BASE：支持base信号；
TUNER_TER_MODE_LITE：支持lite信号；
TUNER_TER_MODE_INVALID：无效值。

B.2.3 事件类型

无。

B.2.4 数据结构

B.2.4.1 有线解调参数

原型：typedef struct {
    int frequency;
    int symbolrate;
    int modulation;
} DVBCTuningParameters_t;
描述：DVBC解调参数。
成员：
frequency：频率；
symbolrate：符号率；
modulation：调制方式。

B.2.4.2 卫星解调参数

原型：typedef struct {
    int frequency;
    int symbolRate;
    TUNER_FE_POLARIZATION_E enPolar;
    int scrambleValue;
} DVBSTuningParameters_t;
描述：DVBS解调参数。
成员：
frequency：下行频率，单位：kHz；
symbolRate：符号率，单位：Symb/s；
enPolar：极化方式；
scrambleValue：物理层扰码初始值，范围0～262141，该值为非0时属于特殊信号，该值只能由信号发送方告知；当频点不是特殊信号时，该值必须配置为默认值0。

B.2.4.3 地面数字电视解调参数

原型：typedef struct {
    int frequency;
    int bandWidth;
    DVB_MOD_TYPE_E enModType;
    unsigned char bReverse;
    TUNER_TER_MODE_E enChannelMode;
    TUNER_TS_PRIORITY_E enDVBTPrio;
} DVBTTuningParameters_t;
描述：DVBT解调参数。
成员：
frequency：频率，单位kHz；
bandWidth：带宽，单位kHz；
enModType：调制方式；

bReverse：频谱翻转处理方式；
enChannelMode：通道中TER模式；
enDVBTPrio：优先级，针对dvb-t。

B.2.4.4 调谐解调参数

```
原型：typedef struct {
    int delivery_type;
    union {
        DvbCTuningParameters_t cable_param;
        DVBSTuningParameters_t sat_param;
        DVBTTuningParameters_t ter_param;
    } tuning_param;
} DvbTuningParameters_t;
```

描述：DVB调谐解调参数
成员：
delivery_type：传输类型；
cable_param：DVBC调谐解调参数；
sat_param：DVBS调谐解调参数；
ter_param：DVBT调谐解调参数。

B.2.4.5 信号状态信息

```
原型：typedef struct {
    int signalQuality;
    int signalStrength;
    int errorRate;
    int signalLevel;
    int signalNoiseRatio;
} DvbSignalStatus_t;
```

描述：信号状态信息。
成员：
signalQuality：信号质量；
signalStrength：信号强度；
errorRate：误码率；
signalLevel：信号电平；
signalNoiseRatio：信噪比。

B.2.5 回调函数定义

无。

B.2.6 接口定义

B.2.6.1 DTVAL_getAllTunerID接口

```
原型：int DTVAL_getAllTunerID(int* ids, int max_count);
```

功能：获取机顶盒的所有 Tuner 的 ID。

参数：ids--输出参数，Tuner 的 ID；

max_count--输入参数，ids 数组的长度，若 Tuner 的数量超过该值，则返回 ERROR_TUNER_MAX_LENGTH。

返回：int，Tuner 的数量。

B.2.6.2 DTVAL_tuneStream接口

原型：int DTVAL_tuneStream(int tuner_id, int original_network_id, int transport_stream_id);

功能：异步方法，将 Tuner 调谐到指定的传送流。

参数：tuner_id--输入参数，Tuner 的 ID，若输入的 Tuner 的 ID 无效，则返回 ERROR_TUNER_INVALID_TUNER_ID；若该 Tuner 所接入的网络中没有该传送流，则返回 ERROR_TUNER_INVALID_TRANSPORT_STREAM；

original_network_id--输入参数，传送流的原始网络 ID；

transport_stream_id--输入参数，传送流的 ID。

返回：int，正常则返回 NO_ERROR，否则返回错误码。

B.2.6.3 DTVAL_tune接口

原型：int DTVAL_tune(int tuner_id, DvbTuningParameters_t param);

功能：异步方法，将 Tuner 调谐到指定的频点。

参数：tuner_id--输入参数，Tuner 的 ID，若输入的 Tuner 的 ID 无效，则返回 ERROR_TUNER_INVALID_TUNER_ID；若频点参数无效，则返回 ERROR_TUNER_INVALID_FREQUENCY；

param--输入参数，调谐频点。

返回：int，正常则返回 NO_ERROR，否则返回错误码。

B.2.6.4 DTVAL_getSystemDeliveryType接口

原型：int DTVAL_getSystemDeliveryType(int tuner_id);

功能：获取Tuner的传送流类型。

参数：tuner_id--输入参数，Tuner 的 ID，若输入的 Tuner 的 ID 无效，则返回 ERROR_TUNER_INVALID_TUNER_ID。

返回：int，Tuner的传送流类型，见DVB_DELIVERY_TYPE_e。

B.2.6.5 DTVAL_getSignalStatus接口

原型：int DTVAL_getSignalStatus(int tuner_id, DvbSignalStatus_t* status);

功能：获取信号状态。

参数：tuner_id--输入参数，Tuner 的 ID，若输入的 Tuner 的 ID 无效，则返回 ERROR_TUNER_INVALID_TUNER_ID。

status：输出参数，信号状态。

返回：int，正常则返回NO_ERROR，否则返回错误码。

B.2.6.6 DTVAL_getTunerStatus接口

原型：int DTVAL_getTunerStatus(int tuner_id);

功能：获取Tuner的锁定状态。

参数：tuner_id--输入参数，Tuner的ID，若输入的Tuner的ID无效，则返ERROR_TUNER_INVALID_TUNER_ID。

返回：int，Tuner的锁定状态，见DVB_TUNER_STATUS_e。

B.2.6.7 DTVAL_getCurrentTunningParam接口

原型：int DTVAL_getCurrentTunningParam(int tunerID, DvbTuningParameters_t *tunningParam);

功能：获取当前调谐参数。

参数：tunerID--输入参数，Tuner的ID；
tunningParam--输出参数,Tuner锁定的频点信息。

返回：成功返回0，失败返回-1。

B.2.6.8 DTVAL_getCurrentTransportStream接口

原型：int DTVAL_getCurrentTransportStream(int tunerId, DVBTS_t* currentStream);

功能：获取当前流信息。

参数：tunerID--输入参数,Tuner标识；
currentStream--输出参数,传输流信息。

返回：成功返回0，失败返回-1。

B.2.6.9 DTVAL_getCurrentService接口

原型：int DTVAL_getCurrentService(int tunerID, DVBService_t* service);

功能：获取当前正在播放的业务。

参数：tunerID--Tuner标识；
service--当前业务。

返回：成功返回0，失败返回-1。

B.2.6.10 DTVAL_setCurrentService接口

原型：void DTVAL_setCurrentService(int tunerID, unsigned short networkID,
unsigned short oriNetworkID, unsigned short tsID, unsigned short serviceID);

功能：设置当前正在播放的业务。

参数：tunerID--Tuner标识。

返回：无。

B.3 节目搜索功能模块

B.3.1 概述

本模块定义了单向广播节目信息搜索功能接口,见表B.3。

表B.3 节目搜索功能模块接口

接口	说明
DTVAL_startScan	开始进行频道搜索。
DTVAL_stopScan	取消频道搜索。

表 B.3（续）

接口	说明
DTVAL_startScanByJson	下载Json数据并解析。
DTVAL_startScan	开始进行频道搜索。
DTVAL_stopScan	取消频道搜索。
DTVAL_startScanByJson	下载Json数据并解析。
DTVAL_JSON_Start	解析Json数据，并保存到数据库中。
DTVAL_JSON_Stop	停止Json数据下载解析。
DTVAL_updateScanResult	更新PSI/SI数据。
DTVAL_saveScanResult	保存PSI/SI数据到NVM。
DTVAL_revertScanResult	从NVM导入PSI/SI数据到RAM中。
DTVAL_deleteScanResult	清除RAM和NVM中的PSI/SI数据。

B.3.2 常量定义

B.3.2.1 搜索状态

原型：typedef enum {

```
    MSG_DVB_SCAN_UNKNOWN = -1,
    MSG_DVB_SCAN_SUCCESS = 10023,
    MSG_DVB_SCAN_START = 10025,
    MSG_DVB_SCAN_FINISHED = 10026,
    MSG_DVB_SCAN_TUNERLOCK_FAILED = 10027,
    MSG_DVB_SCAN_PAT_FAILED = 10108,
    MSG_DVB_SCAN_PMT_FAILED = 10109,
    MSG_DVB_SCAN_NIT_FAILED = 10110,
    MSG_DVB_SCAN_NIT_SUCCESS = 10024,
    MSG_DVB_SCAN_FIND_SERVICES = 10028,
    MSG_DVB_SCAN_STOP_SUCCESS = 10029,
    MSG_DVB_SCAN_STOP_FAILED = 10030,
    MSG_DVB_SCAN_SAVE_SUCCESS = 10101,
    MSG_DVB_SCAN_SAVE_FAILED = 10102,
    MSG_DVB_SCAN_REVERT_SUCCESS = 10103,
    MSG_DVB_SCAN_REVERT_FAILED = 10104,
    MSG_DVB_SCAN_DELETE_SUCCESS = 10105,
    MSG_DVB_SCAN_DELETE_FAILED = 10106,
    MSG_DVB_SCAN_TUNER_SETDONE = 10107
} MSG_DVB_SCAN_STATUS_e;
```

描述：搜索过程事件信息。

成员：

MSG_DVB_SCAN_UNKNOWN：未知状态；

MSG_DVB_SCAN_SUCCESS：搜索成功；

MSG_DVB_SCAN_START：搜索开始；
MSG_DVB_SCAN_FINISHED：搜索完成；
MSG_DVB_SCAN_TUNERLOCK_FAILED：搜索锁频失败；
MSG_DVB_SCAN_PAT_FAILED：搜索PAT失败；
MSG_DVB_SCAN_PMT_FAILED：搜索PMT失败；
MSG_DVB_SCAN_NIT_FAILED：搜索NIT失败；
MSG_DVB_SCAN_NIT_SUCCESS：NIT搜索成功；
MSG_DVB_SCAN_FIND_SERVICES：搜索到节目；
MSG_DVB_SCAN_STOP_SUCCESS：搜索停止成功；
MSG_DVB_SCAN_STOP_FAILED：搜索停止失败；
MSG_DVB_SCAN_SAVE_SUCCESS：数据保存成功；
MSG_DVB_SCAN_SAVE_FAILED：数据保存失败；
MSG_DVB_SCAN_REVERT_SUCCESS：数据转存成功；
MSG_DVB_SCAN_REVERT_FAILED：数据转存失败；
MSG_DVB_SCAN_DELETE_SUCCESS：数据删除成功；
MSG_DVB_SCAN_DELETE_FAILED：数据删除失败；
MSG_DVB_SCAN_TUNER_SETDONE：锁频成功。

B.3.3 事件类型

无。

B.3.4 数据结构

B.3.4.1 搜索结果

原型：typedef struct {
    int tsCount;
    int serviceCount;
} DVBScanResultCount_t;
描述：搜索结果。
成员：
tsCount：搜索到的传输流个数；
serviceCount：搜索到的节目个数。

B.3.5 回调函数定义

B.3.5.1 DTVAL_scanNotify回调

原型：typedef void (*DTVAL_scanNotify)(void* env, int scanstatuscode,
      unsigned int frequency, int cb_priv, void* pChainIndex,
      int numofChainIndex);
功能：频道扫描回调函数。
参数：env--TVOS-J环境变量；
      scanstatuscode--搜索过程状态值，取值见MSG_DVB_SCAN_STATUS_e；
      frequency--频点；

cb_priv--对象指针（TVOS-H使用）；

pChainIndex--链表指针（TVOS-J使用），用于返回搜索的结果。

TVOS-J只使用MSG_DVB_SCAN_STATUS_e中的四个选项：

——MSG_DVB_SCAN_NIT_SUCCESS：当NIT搜索完成后进行回调，此时第二个参数频点等于0，第四个参数void*为DVBTS_t*，即返回的是搜索NIT搜索到的所有TS流的结构体链表头指针。

——MSG_DVB_SCAN_SUCCESS：当搜索某个频点完成后进行回调，此时第二个参数为该频点，第四个参数void*为DVBService_t*，即返回的是搜索到的所有service的结构体链表头指针。

——MSG_DVB_SCAN_FAILED：当搜索某个频点失败后进行回调，此时第二个参数为该频点，第四个参数为空。

——MSG_DVB_SCAN_FINISHED：当整体搜索（自动搜索、手动搜索、区间搜索）完成后进行回调，此时第二个参数为0，第四个参数void*为DVBScanResultCount_t*，即返回本次搜索共搜索到的TS流数量和Service数量。

numofChainIndex--描述pChainIndex指针个数或空间。

返回：无。

B.3.6 接口定义

B.3.6.1 DTVAL_startScan接口

原型：void DTVAL_startScan(void* env, int type, int tunerID, DvbTuningParameters_t* params, int params_count, DTVAL_scanNotify cb, int cb_priv);

功能：开始进行频道搜索,同步方法。

参数：env--TVOS-J环境变量；

type--表示搜索类型；

tunerID--表示从哪个Tuner搜索；

params--表示调谐参数；

--type=0,手动搜索，数组的长度为1；

--type=1，自动搜索，数组的长度等于运营商部署的起始频点数量；

--type=2，区间搜索，数组的长度为2，params[0]表示开始搜索频点的调谐参数，params[1]表示终止搜索频点的调谐参数。

params_count--表示调谐参数数量；

cb--回调函数；

cb_priv--回调函数参数，对象指针。

返回：无。

B.3.6.2 DTVAL_stopScan接口

原型：void DTVAL_stopScan(int tunerid);

功能：取消搜索。

参数：tunerid--表示从哪个Tuner搜索。

返回：无。

B.3.6.3 DTVAL_startScanByJson接口

原型：void DTVAL_startScanByJson(void* env,int type,int tunerID,DvbTuningParameters_t* params,int params_count,DTVAL_scanNotify cb,int cb_priv,int pid,int tableid);

功能：下载json数据并解析。Tuner锁定rpstru_ScanParam参数的频点，下载对应pid、tableid下的数据，根据描述信息，dtv内部再下载对应的json数据，并解析json数据保存到DB中。

参数：env--TVOS-J环境变量；
type--表示搜索类型；
tunerID--保存对应的TunerID；
params--节目信息数据所在频点的调谐解调参数；
cb--回调函数；
cb_priv--回调函数参数，对象指针；
pid--节目信息数据所在TS包的PID；
tableid--节目信息数据分配的tableid。

返回：无

B.3.6.4 DTVAL_JSON_Start接口

原型：int DTVAL_JSON_Start(int tunerID, char* pJson, int nLen);

描述：解析pJson数据，并保存到数据库中。

参数：tunerID--主要用于指定保存到哪个数据库；
pJson--节目信息数据；
nLen--长度。

返回：int，返回错误码。

B.3.6.5 DTVAL_JSON_Stop接口

原型：int DTVAL_JSON_Stop(int tunerID);

描述：停止数据下载解析。

参数：tunerID—表示停止哪个Tuner的搜索。

返回：int，返回错误码。

B.3.6.6 DTVAL_updateScanResult接口

原型：int DTVAL_updateScanResult(int tunerID);

功能：更新PSI/SI数据。

参数：tunerID--表示更新哪个Tuner的搜索结果。

返回：int，成功返回0，失败返回-1。

B.3.6.7 DTVAL_saveScanResult接口

原型：int DTVAL_saveScanResult(int tunerID);

功能：保存PSI/SI数据到NVM。在搜索完毕后，调用此方法更新NVM中的数据，否则重启接收终端后还将保持原有的频道数据。

参数：tunerID--表示保存哪个Tuner的搜索结果。

返回：int，成功返回0，失败返回-1。

B.3.6.8 DTVAL_revertScanResult接口

原型：int DTVAL_revertScanResult(int tunerID);
功能：从NVM导入PSI/SI数据到RAM中。
参数：tunerID--表示导入哪个Tuner的搜索结果。
返回：int，成功返回0，失败返回-1。

B.3.6.9 DTVAL_deleteScanResult接口

原型：int DTVAL_deleteScanResult(int tunerID);
功能：清除RAM 和NVM 中的PSI/SI 数据。
参数：tunerID--表示清除哪个Tuner的搜索结果。
返回：int，成功返回0，失败返回-1。

B.4 广播协议信息查询与数据过滤功能模块

B.4.1 概述

定义了从传输流中查询相关信息和过滤通过DVB标准传输的各种数据的功能接口，见表B.4。

表B.4 广播协议信息查询与数据过滤功能模块接口

接口	说明
DTVAL_getUnusedFilterNumber	获得系统中当前未使用的段过滤器数量。
DTVAL_requestFilter	申请使用一个段过滤器。
DTVAL_releaseFilter	释放占用的段过滤器。
DTVAL_attachStream	将段过滤器与传送流绑定。
DTVAL_detachStream	将段过滤器与其绑定的传送流断开。
DTVAL_startFiltering	开始过滤。
DTVAL_stopFiltering	停止过滤。
DTVAL_startTableMonitor	启动表格更新的监控。
DTVAL_stopTableMonitor	停止对表格更新事件的监控。
DTVAL_getSIInfo	获取SI信息。
DTVAL_releaseSIInfo	释放SI信息。
DTVAL_retrieveActualNetwork	从现行NIT表中获取当前网络信息。
DTVAL_retrieveActualTransportStreams	从现行NIT表中获取当前网络的所有传送流信息。
DTVAL_releaseDVBService	释放DVBService_t*结构体。
DTVAL_releaseDVBTS	释放DVBTS_t* 结构体。
DTVAL_retrieveActualServices	从当前传输流中获取所有业务信息。
DTVAL_retrievePMTService	获取某个业务相关的PMT描述的业务信息。
DTVAL_retrieveTimeFromTDT	从当前传送流承载的TDT表中获取时间信息。
DTVAL_retrieveTimeFromTOT	从当前传送流承载的TOT表中获取时间信息。

B.4.2 常量定义

B.4.2.1 常用PID

原型：#define PAT_PID 0x0000
描述：PSI/SI PAT PID
原型：#define CAT_PID 0x0001
描述：PSI/SI CAT PID
原型：#define TSDT_PID 0x0002
描述：PSI/SI TSDT PID
原型：#define NIT_PID 0x0010
描述：PSI/SI NIT PID
原型：#define SDT_PID 0x0011
描述：PSI/SI SDT PID
原型：#define BAT_PID 0x0011
描述：PSI/SI BAT PID
原型：#define EIT_PID 0x0012
描述：PSI/SI EIT PID
原型：#define RST_PID 0x0013
描述：PSI/SI RST PID
原型：#define TDT_PID 0x0014
描述：PSI/SI TDT PID
原型：#define TOT_PID 0x0014
描述：PSI/SI TOT PID
原型：#define DIT_PID 0x001E
描述：PSI/SI DIT PID
原型：#define SIT_PID 0x001F
描述：PSI/SI SIT PID

B.4.2.2 常用TableID

原型：#define PAT_TABLE_ID 0x00
描述：PSI/SI PAT TABLEID
原型：#define CAT_TABLE_ID 0x01
描述：PSI/SI CAT TABLEID
原型：#define PMT_TABLE_ID 0x02
描述：PSI/SI PMT TABLEID
原型：#define TSDT_TABLE_ID 0x03
描述：PSI/SI TSDT TABLEID
原型：#define NIT_TABLE_ID 0x40
描述：PSI/SI NIT TABLEID
原型：#define NIT_OTHER_TABLE_ID 0x41
描述：PSI/SI NIT_OTHER TABLEID
原型：#define SDT_TABLE_ID 0x42
描述：PSI/SI SDT TABLEID
原型：#define SDT_OTHER_TABLE_ID 0x46
描述：PSI/SI SDT_OTHER TABLEID

原型：#define BAT_TABLE_ID 0x4A
描述：PSI/SI BAT TABLEID
原型：#define EIT_TABLE_ID 0x4E
描述：PSI/SI EIT TABLEID
原型：#define EIT_OTHER_TABLE_ID 0x4F
描述：PSI/SI EIT_OTHER TABLEID
原型：#define TDT_TABLE_ID 0x70
描述：PSI/SI TDT TABLEID
原型：#define RST_TABLE_ID 0x71
描述：PSI/SI RST TABLEID
原型：#define TOT_TABLE_ID 0x73
描述：PSI/SI TOT TABLEID
原型：#define DIT_TABLE_ID 0x7E
描述：PSI/SI DIT TABLEID
原型：#define SIT_TABLE_ID 0x7F
描述：PSI/SI SIT TABLEID

B.4.2.3 表格类型

```
原型：enum SI_TABLE_TYPE_E {
    SI_TABLE_TYPE_UNKNOWN = 0,
    SI_TABLE_TYPE_BAT = 1,
    SI_TABLE_TYPE_NIT = 2,
    SI_TABLE_TYPE_PAT = 3,
    SI_TABLE_TYPE_PMT = 4,
    SI_TABLE_TYPE_SDT = 5,
    SI_TABLE_TYPE_EIT = 6,
    SI_TABLE_TYPE_EIT_PF = 7,
    SI_TABLE_TYPE_EIT_SCHEDULE = 8,
    SI_TABLE_TYPE_TDT = 9,
    SI_TABLE_TYPE_TOT = 10,
    SI_TABLE_TYPE_DATA = 11
};
```

描述：标准PSI/SI 表格。
成员：
SI_TABLE_TYPE_UNKNOWN：无效表格；
SI_TABLE_TYPE_BAT：BAT表格；
SI_TABLE_TYPE_NIT：NIT表格；
SI_TABLE_TYPE_PAT：PAT表格；
SI_TABLE_TYPE_PMT：PMT表格；
SI_TABLE_TYPE_SDT：SDT表格；
SI_TABLE_TYPE_EIT：EIT表格；
SI_TABLE_TYPE_EIT_PF：EIT_PF表格；

SI_TABLE_TYPE_EIT_SCHEDULE：EIT_SCHEDULE表格；
SI_TABLE_TYPE_TDT：TDT表格；
SI_TABLE_TYPE_TOT：TOT表格；
SI_TABLE_TYPE_DATA：私有数据。

B.4.2.4 节目过滤类型

原型：#define FILTER_TYPE_SERVICETYPE 1000
描述：依据类型过滤节目
原型：#define FILTER_TYPE_FAV 1001
描述：依据喜好过滤节目
原型：#define FILTER_TYPE_BAT 1002
描述：依据BAT过滤节目
原型：#define FILTER_TYPE_FTA 1003
描述：依据FTA过滤节目
原型：#define FILTER_TYPE_HIDE 1004
描述：依据隐藏过滤节目

B.4.2.5 排序方法

原型：#define NGB_DVB_SORT_TYPE_NETWORK_ID 0x01
描述：依据network_id排序。
原型：#define NGB_DVB_SORT_TYPE_BOUQUET_ID 0x02
描述：依据bouquet_id排序。
原型：#define NGB_DVB_SORT_TYPE_ONET_ID 0x03
描述：依据original_network_id排序。
原型：#define NGB_DVB_SORT_TYPE_TS_ID 0x04
描述：依据transport_stream_id排序。
原型：#define NGB_DVB_SORT_TYPE_SERVICE_ID 0x05
描述：依据service_id排序。
原型：#define NGB_DVB_SORT_TYPE_SERVICE_TYPE 0x06
描述：依据service_type排序。
原型：#define NGB_DVB_SORT_TYPE_SERVICE_NAME 0x07
描述：依据service_name排序。
原型：#define NGB_DVB_SORT_TYPE_CHANNEL_ID 0x10
描述：依据用户频道号排序。
原型：#define NGB_DVB_SORT_TYPE_LOGICAL_ID 0x11
描述：依据逻辑频道号排序。
原型：#define NGB_DVB_SORT_TYPE_FTA_SCR 0x20
描述：依据是否加密排序。
原型：#define NGB_DVB_SORT_ORDER_NON 0x00
描述：不进行排列。
原型：#define NGB_DVB_SORT_ORDER_ASC 0x01

描述：若排序依据为DVB_SORT_TYPE_FTA_SCR，则表示非加密到加密排列，若排序依据为其它，则表示升序排列。

原型：#define NGB_DVB_SORT_ORDER_DESC 0x02

描述：若排序依据为DVB_SORT_TYPE_FTA_SCR，则表示加密到非加密排列，若排序依据为其它，则表示降序排列

B.4.2.6 业务类型

原型：#define NGB_DVB_SERVICE_TYPE_DTV 0x01

描述：数字电视广播业务

原型：#define NGB_DVB_SERVICE_TYPE_DAB 0x02

描述：数字声音广播业务

原型：#define NGB_DVB_SERVICE_TYPE_TELETEXT 0x03

描述：图文电视业务

原型：#define NGB_DVB_SERVICE_TYPE_NVOD_REF 0x04

描述：NVOD 参考业务

原型：#define NGB_DVB_SERVICE_TYPE_NVOD_SHIFT 0x05

描述：NVOD 时移业务

原型：#define NGB_DVB_SERVICE_TYPE_MOSAIC 0x06

描述：马赛克业务

原型：#define NGB_DVB_SERVICE_TYPE_DATA 0x0c

描述：数据广播业务

原型：#define MAX_FILTER_ATTACH 33

描述：最大过滤器数量

原型：#define SI_MAX_NETOWRK_NAME_LE 64

描述：最大网络名称长度

原型：#define SI_MAX_BOUQUET_NAME_LEN 64

描述：最大分组名称长度

原型：#define MAX_NAME_LENGTH 256

描述：最大名称长度

原型：#define SI_MAX_SERVICE_STREAMS_NU 20

描述：节目最大基本流个数

原型：#define MAX_BOUQUET_GRP_NUM 10

描述：节目最多分组数

原型：#define MAX_SHIFT_SERVICES_NUM 100

描述：时移参考节目的最大传输节目个数

原型：#define MAX_SCAN_LENGTH 100

描述：搜索最多频点个数

原型#define SI_MAX_FILTER_CAPTURE_NUM 64

描述：最大filter个数

B.4.2.7 排序类型

原型：enum {

```
    SI_INFO_SORT_NONE = 0,
    SI_INFO_SORT_ASC = 1,
    SI_INFO_SORT_DESC = 2
};
```

描述：排序类型。

成员：

SI_INFO_SORT_NONE：不排序；

SI_INFO_SORT_ASC：升序；

SI_INFO_SORT_DESC：降序。

B.4.2.8 信息类型

```
原型：typedef enum {
    SI_INFO_UNKNOWN = 0,
    SI_INFO_NETWORK,
    SI_INFO_BOUQUET,
    SI_INFO_TS,
    SI_INFO_ES,
    SI_INFO_SERVICE,
    SI_INFO_EVENT,
    SI_INFO_MOSAIC,
    SI_INFO_TIME,
} DVB_SI_INFO_TYPE_E;
```

描述：SI 信息类型。

成员：

SI_INFO_UNKNOWN：未知类型；

SI_INFO_NETWORK：SI信息结构体为DVBNetwork_t；

SI_INFO_BOUQUET：SI信息结构体为DVBBouquet_t；

SI_INFO_TS：SI信息结构体为DVBTS_t；

SI_INFO_ES：SI信息结构体为DVBElementaryStream_t；

SI_INFO_SERVICE：SI信息结构体为DVBService_t；

SI_INFO_EVENT：SI信息结构体为DVBEvent_t；

SI_INFO_MOSAIC：SI信息结构体为DVBMosaic_t；

SI_INFO_TIME：预留。

B.4.2.9 信息过滤屏蔽类型

```
原型：typedef enum {
    SI_INFO_NO_MASK = 0,
    SI_INFO_NETWORK_ID_MASK = 0x1,
    SI_INFO_ON_ID_MASK = (0x1 << 1),
    SI_INFO_TRANSPORT_STREAM_ID_MASK = (0x1 << 2),
    SI_INFO_SERVICE_ID_MASK = (0x1 << 3),
    SI_INFO_BOUQUET_ID_MASK = (0x1 << 4),
```

```
    SI_INFO_EVENT_ID_MASK = (0x1 << 5),
    SI_INFO_START_DATE_MASK = (0x1 << 6),
    SI_INFO_START_TIME_MASK = (0x1 << 7),
    SI_INFO_END_DATE_MASK = (0x1 << 8),
    SI_INFO_END_TIME_MASK = (0x1 << 9),
    SI_INFO_SERVICE_TYPE_MASK = (0x1 << 10),
    SI_INFO_EVENT_TABLE_MASK = (0x1 << 11),
    SI_INFO_EVENT_DIRECTION_MASK = (0x1 << 12),
    SI_INFO_ELEMENTARY_STREAM_PID_MASK = (0x1 << 13),
} DVB_SI_INFO_MASK_E;
```

描述：SI 信息Mask。

成员：

SI_INFO_NO_MASK ：不匹配过滤；

SI_INFO_NETWORK_ID_MASK ：匹配networkID；

SI_INFO_ON_ID_MASK：匹配original network id；

SI_INFO_TRANSPORT_STREAM_ID_MASK：匹配transport stream id；

SI_INFO_SERVICE_ID_MASK：匹配service id；

SI_INFO_BOUQUET_ID_MASK：匹配bouquet id；

SI_INFO_EVENT_ID_MASK：匹配event id；

SI_INFO_START_DATE_MASK：匹配event 的起始日期，获取大于起始日期的节目；

SI_INFO_START_TIME_MASK：匹配event 的起始时间，获取大于起始时间的节目；

SI_INFO_END_DATE_MASK：匹配event 的结束日期，获取小于起始日期的节目；

SI_INFO_END_TIME_MASK：匹配event 的结束时间，获取小于起始时间的节目；

SI_INFO_SERVICE_TYPE_MASK：匹配节目的类型；

SI_INFO_EVENT_TABLE_MASK：匹配所需要的event类型，分为present/following/schedule；

SI_INFO_EVENT_DIRECTION_MASK：匹配event查询方向，向前或向后；

SI_INFO_ELEMENTARY_STREAM_PID_MASK：匹配PID。

B.4.2.10 信息排序类型

```
原型：typedef enum {
    SI_INFO_SORT_TYPE_NONE = 0
    SI_INFO_SORT_TYPE_NETWORK_ID = 0x01,
    SI_INFO_SORT_TYPE_BOUQUET_ID = 0x02,
    SI_INFO_SORT_TYPE_ONET_ID = 0x03,
    SI_INFO_SORT_TYPE_TRANSPORT_STREAM_ID = 0x04,
    SI_INFO_SORT_TYPE_SERVICE_ID = 0x05,
    SI_INFO_SORT_TYPE_SERVICE_TYPE = 0x06,
    SI_INFO_SORT_TYPE_EVENT_TYPE = 0x07,
    SI_INFO_SORT_TYPE_START_DATE_TIME = 0x08
} SI_INFO_SORT_TYPE_E;
```

描述：信息排序类型。

成员：

SI_INFO_SORT_TYPE_NONE：不排序；
SI_INFO_SORT_TYPE_NETWORK_ID：以network ID排序；
SI_INFO_SORT_TYPE_BOUQUET_ID：以bouquet ID排序；
SI_INFO_SORT_TYPE_ONET_ID：以original network ID排序；
SI_INFO_SORT_TYPE_TRANSPORT_STREAM_ID：以transport stream ID排序；
SI_INFO_SORT_TYPE_SERVICE_ID：以service ID排序；
SI_INFO_SORT_TYPE_SERVICE_TYPE：以service type排序；
SI_INFO_SORT_TYPE_EVENT_TYPE：以event type排序；
SI_INFO_SORT_TYPE_START_DATE_TIME：以event的起始日期和时间排序。

B.4.3 事件类型

无。

B.4.4 数据结构

B.4.4.1 绑定传输流信息

```
原型：typedef struct {
    int filterID;
    int tunerID;
    int transportStreamID;
} DVBAttach_stream_t;
```

描述：绑定传输流信息。
成员：
filterID：过滤器编号；
tunerID：调谐解调器编号；
transportStreamID：传输流ID。

B.4.4.2 段数据结构

```
原型：typedef struct _dvb_section {
    unsigned char section_number;
    unsigned char *section_buffer_ptr;
    unsigned short section_buffer_length;
    struct _dvb_section *next;
} DVBSection_t;
```

描述：Section数据结构。
成员：
section_number：当前secion编号；
section_buffer_ptr：数据指针；
section_buffer_length：数据长度，字节；
next：下一个数据section指针。

B.4.4.3 表数据结构

```
原型：typedef struct _si_table {
```

```
    unsigned short pid;
    unsigned short table_id;
    int num_of_sections;
    DVBSection_t *sections;
} DVBTable_t;
```

描述：Table数据结构。

成员：

pid：数据PID；

table_id：数据的TableID；

num_of_sections：总section数量；

sections：section数据结构指针。

B.4.4.4 描述子结构

```
原型：typedef struct _si_descriptor {
    unsigned char table_id;
    unsigned char descriptor_tag;
    unsigned char *descriptor_buffer_ptr;
    unsigned short descriptor_buffer_length;
    struct _si_descriptor *next;
} DVBDescriptor_t;
```

描述：描述子信息。

成员：

table_id：数据的TableID；

descriptor_tag：描述子tag；

descriptor_buffer_ptr：描述子buffer地址；

descriptor_buffer_length：描述子buffer长度；

next：下一个tag。

B.4.4.5 表格过滤参数

```
原型：typedef struct {
    unsigned short pid;
    int timeout;
    unsigned char mask;
    unsigned char tableID;
    unsigned short tableExt;
    unsigned short tableExt2;
    unsigned short tableExt3;
} DVBSi_get_table_config_t;
```

描述：表格过滤参数。

成员：

pid：数据PID；

timeout：超时时间（单位：毫秒），值>0；

mask：过滤条件屏蔽字；
tableID：过滤tableID，对应section[0]；
tableExt：过滤扩展table ID，对应section[3] & section[4]；
tableExt2：过滤扩展table ID 2，对应section[8] & section[9]；
tableExt3：过滤扩展table ID 3，对应section[10] & section[11]。

B.4.4.6 过滤描述子参数

原型：typedef struct {
    unsigned short pid;
    int timeout;
    unsigned char mask;
    unsigned char tableID;
    unsigned char descTag;
    unsigned short tableExt;
    unsigned short tableExt2;
    unsigned short tableExt3;
} DVBSi_get_descriptor_config_t;
描述：描述子配置信息。
成员：
pid：过滤PID；
timeout：超时时间，单位：毫秒（取值范围为≥0）；
mask：过滤条件屏蔽字；
tableID：过滤tableID，对应section[0]；
descTag：描述子标识；
tableExt：过滤扩展table ID，对应section[3] & section[4]；
tableExt2：过滤扩展table ID 2，对应section[8] & section[9]；
tableExt3：过滤扩展table ID 3，对应section[10] & section[11]。

B.4.4.7 节目定位符

原型：typedef struct __NGB_SERVICELOCATOR__ {
    int ri_TsID;
    int ri_NetworkId;
    int ri_OrgNetId;
    int ri_ServiceID;
} NGB_SERVICELOCATOR;
描述：节目信息。
成员：
ri_TsID：传输流ID；
ri_NetworkId：网络ID；
ri_OrgNetId：原始网络ID；
ri_ServiceID：节目ID。

B.4.4.8 网络信息

原型：typedef struct {
    unsigned short network_id;
    unsigned char delivery_type;
    char network_name[SI_MAX_NETOWRK_NAME_LEN];
    char short_network_name[SI_MAX_NETOWRK_NAME_LEN];
    DVBDescriptor_t* descriptors;
} DVBNetwork_t;
描述：网络信息。
成员：
network_id：网络ID；
delivery_type：传输类型；
network_name[SI_MAX_NETOWRK_NAME_LEN]：网络名称（长）；
short_network_name[SI_MAX_NETOWRK_NAME_LEN]：网络名称（短）；
descriptors：描述信息。

B.4.4.9 传输流信息

原型：typedef struct {
    unsigned short network_id;
    unsigned short original_network_id;
    unsigned short transport_stream_id;
    int bouquet_id;
    int delivery_type;
    union {
            DVBCTuningParameters_t cableParam;
            DVBSTuningParameters_t sat_param;
            DVBTTuningParameters_t ter_param;
    } deliveryParameter;
    DVBDescriptor_t* descriptors;
} DVBTS_t;
描述：传输流信息。
成员：
network_id：网络ID；
original_network_id：原始网络ID；
transport_stream_id：传输流ID；
bouquet_id：bouquetID；
delivery_type：传输类型；
cableParam：DVBC参数；
sat_param：DVBS参数；
ter_param：DVBT参数；
descriptors：描述信息。

B.4.4.10 条件接收信息

```
原型：typedef struct {
    int cas_id;
    int ecm_pid;
    int emm_pid;
} DVBCA_info_t;
```

描述：CA信息。

成员：

cas_id：CA System ID；

ecm_pid：ECM PID；

emm_pid：EMM PID。

B.4.4.11 节目分组信息

```
原型：typedef struct {
    unsigned short network_id;
    unsigned short bouquet_id;
    char bouquet_name[SI_MAX_BOUQUET_NAME_LEN];
    char short_bouquet_name[SI_MAX_BOUQUET_NAME_LEN];
    DVBDescriptor_t* descriptors;
} DVBBouquet_t;
```

描述：节目分组信息。

成员：

network_id：网络ID；

bouquet_id：BouquetID；

bouquet_name[SI_MAX_BOUQUET_NAME_LEN]：BouquetName（长）；

short_bouquet_name[SI_MAX_BOUQUET_NAME_LEN]：BouquetName（短）；

descriptors：描述信息。

B.4.4.12 基本流信息定义

```
原型：typedef struct {
    unsigned char stream_type;
    unsigned short elem_pid;
    unsigned char component_tag;
    int data_broadcast_ID;
    int association_tag;
    unsigned short network_id;
    unsigned short original_network_id;
    unsigned short transport_stream_id;
    unsigned short service_id;
    char lingual[4];
    int bitrate;
    int width;
    int height;
```

```
    int DVBCA_info_num;
    DVBCA_info_t stru_CaData[8];
    DVBDescriptor_t* descriptors;
} DVBElementaryStream_t;
```

描述：基本流信息。

成员：

stream_type：基本流类型；

elem_pid：基本流PID；

component_tag：基本流组件标签；

data_broadcast_ID：数据广播ID；

association_tag：关联描述子标识；

network_id：网络标识；

original_network_id：原始网络标识；

transport_stream_id：传输流标识；

service_id：业务标识；

lingual[4]：语种，适用于音频和图文；

bitrate：视频传输速率，单位：bps；

width：视频宽度：pixel，仅适用于视频；

height：高度：pixel，仅适用于音频；

DVBCA_info_num：条件接收信息个数；

stru_CaData[8]：条件接收信息；

descriptors：描述符。

B.4.4.13 节目信息

```
原型：typedef struct {
    unsigned short network_id;
    unsigned short original_network_id;
    unsigned short transport_stream_id;
    unsigned short service_id;
    unsigned char EIT_sch_flag;
    unsigned char EIT_pf_flag;
    unsigned char running_status;
    unsigned char free_ca_mode;
    unsigned short pcr_pid;
    unsigned short logic_channel_number;
    unsigned char service_type;
    int stream_count;
    int current_vstream;
    int current_astream;
    DVBElementaryStream_t streams[SI_MAX_SERVICE_STREAMS_NUM];
    char service_provider_name[MAX_NAME_LENGTH];
    char short_service_provider_name[MAX_NAME_LENGTH];
```

```
    char service_name[MAX_NAME_LENGTH];
    char short_service_name[MAX_NAME_LENGTH];
    unsigned short time_ref_service_id;
    int time_shift_service_count;
    unsigned short time_shift_service_ids[MAX_SHIFT_SERVICES_NUM];
    int bouquet_ids_count;
    unsigned short bouquet_ids[MAX_BOUQUET_GRP_NUM];
    DVBDescriptor_t* descriptors;
} DVBService_t;
```

描述：节目信息。

成员：

network_id：网络id；

original_network_id：原始网络id；

transport_stream_id：传输流 id；

service_id：节目id；

EIT_sch_flag：EIT sch flag；

EIT_pf_flag：eit present following flag；

running_status：运行状态；

free_ca_mode：加密或清流标识；

pcr_pid：service 的pcr pid；

logic_channel_number：逻辑频道号BAT；

service_type：节目类型；

stream_count：service 里面stream 数量；

current_vstream：当前的视频stream 索引号，在streams中的索引；

current_astream：当前的音频stream 索引号，在streams中的索引；

streams[SI_MAX_SERVICE_STREAMS_NUM]：service里面的所有stream；

service_provider_name[MAX_NAME_LENGTH]：service provider name；

short_service_provider_name[MAX_NAME_LENGTH]：short service provider name；

service_name[MAX_NAME_LENGTH]：service name；

short_service_name[MAX_NAME_LENGTH]：short service name；

time_ref_service_id：only once exist time_ref_service_id time shifted service descriptorr；

time_shift_service_count：包含多少个时移service ；

time_shift_service_ids[MAX_SHIFT_SERVICES_NUM]：时移servicelist；

bouquet_ids_count：BouquetID 个数；

short bouquet_ids[MAX_BOUQUET_GRP_NUM]：属于多少个bouquet组；

descriptors：描述符。

B.4.4.14 调制信息

```
原型：typedef struct {
    int i_CarrierIndex;
    int i_TsID;
    int i_NetID;
```

```
    int i_OrgNetId;
    int i_FrequencyKHz;
    int i_SymbolRateKb;
    int i_QAM;
} DVBCarrierInfo_t;
```

描述：调制信息。

成员：

i_CarrierIndex：所在的频点索引；

i_TsID：传输流ID；

i_NetID：网络ID；

i_OrgNetId：原始网络ID；

i_FrequencyKHz：频点；

i_SymbolRateKb：符号率；

i_QAM：QAM。

B.4.4.15 网络匹配信息

```
原型：typedef struct {
    unsigned short m_networkID;
} DVBNetwork_match_t;
```

描述：网络匹配信息。

成员：

m_networkID：网络ID。

B.4.4.16 业务分群匹配信息

```
原型：typedef struct {
    unsigned short m_networkID;
    unsigned short m_originalNetworkID;
    unsigned short m_transportStreamID;
    unsigned short m_serviceID;
    unsigned short m_bouquetID;
} DVBBouquet_match_t;
```

描述：业务分群匹配信息。

成员：

m_networkID：网络ID；

m_originalNetworkID：原始网络ID；

m_transportStreamID：传输流ID；

m_serviceID：节目ID；

m_bouquetID：业务群ID。

B.4.4.17 传输流匹配信息

```
原型：typedef struct {
    unsigned short m_networkID;
```

```
    unsigned short m_originalNetworkID;
    unsigned short m_transportStreamID;
    unsigned short m_bouquetID;
} DVBTransport_stream_match_t;
```

描述：传输流匹配信息。

成员：

m_networkID：网络ID；

m_originalNetworkID：原始网络ID；

m_transportStreamID：传输流ID；

m_bouquetID：业务群ID。

B.4.4.18 业务匹配信息

```
原型：typedef struct {
    unsigned short m_networkID;
    unsigned short m_originalNetworkID;
    unsigned short m_transportStreamID;
    unsigned short m_serviceID;
    unsigned short m_bouquetID;
    unsigned short m_serviceType;
} DVBService_match_t;
```

描述：业务匹配信息。

成员：

m_networkID：网络ID；

m_originalNetworkID：原始网络ID；

m_transportStreamID：传输流ID；

m_serviceID：业务ID；

m_bouquetID：业务群ID；

m_serviceType：业务类型。

B.4.4.19 事件匹配信息

```
原型：typedef struct {
    unsigned short m_networkID;
    unsigned short m_originalNetworkID;
    unsigned short m_transportStreamID;
    unsigned short m_serviceID;
    unsigned short m_eventID;
    DVBDateTime_t m_startTime;
    DVBDateTime_t m_endTime;
    char m_eventTable;
    unsigned short m_eventContentType;
} DVBEvent_match_t;
```

描述：事件匹配信息。

成员：
m_networkID：网络ID；
m_originalNetworkID：原始网络ID；
m_transportStreamID：传输流ID；
m_serviceID：业务ID；
m_eventID：节目ID；
m_startTime：节目起始时间，bcd格式；
m_endTime：节目结束时间，bcd格式；
m_eventTable：节目table，取值代表prsent 0x00，following 0x01，schedule 0x02；
m_eventContentType：节目内容类型。

B.4.4.20 基本流匹配信息

```
原型：typedef struct {
    unsigned short m_networkID;
    unsigned short m_originalNetworkID;
    unsigned short m_transportStreamID;
    unsigned short m_serviceID;
    unsigned short m_elementaryStreamID;
} DVBDescriptor_match_t;
```

描述：基本流匹配信息。
成员：
m_networkID：网络ID；
m_originalNetworkID：原始网络ID；
m_transportStreamID：传输流ID；
m_serviceID：业务ID；
m_elementaryStreamID：基本流ID。

B.4.4.21 匹配信息

```
原型：typedef struct {
    int m_type;
    int m_mask;
    union {
        DVBNetwork_match_t m_network;
        DVBBouquet_match_t m_bouquet;
        DVBTransport_stream_match_t m_transportStream;
        DVBService_match_t m_service;
        DVBEvent_match_t m_event;
    } m_match;
} DVBInfo_match_t;
```

描述：SI 匹配信息。
成员：
m_type：要过滤的type，参考DVB_SI_INFO_TYPE_E；

m_mask：要过滤的mask，参考DVB_SI_INFO_MASK_E；
m_match：要匹配的内容。

B.4.4.22 排序信息

原型：typedef struct {
    int num;
    int sortType[MAX_SORT_NUM];
    int sortOrder[MAX_SORT_NUM];
} DVBInfo_sortorder_t;
描述：排序信息。
成员：
num：排序基准数量；
sortType[MAX_SORT_NUM]：排序依据；
sortOrder[MAX_SORT_NUM]：排序顺序。

B.4.5 回调函数定义

B.4.5.1 DTVAL_reportFilteredSection回调

原型：typedef void (*DTVAL_reportFilteredSection)(void* data, int filter_id,
      char* section_data, int section_length);
功能：PSI/SI section过滤。
参数：data--传入的附加参数；
      filter_id--输入参数，上报数据的过滤器的ID；
      section_data--输入参数，过滤到的段数据；
      section_length--输入参数，过滤到的段数据的长度。
返回：无。

B.4.5.2 DTVAL_reportFilterForceDisconnected回调

原型：typedef void (*DTVAL_reportFilterForceDisconnected)(void* data, int filter_id);
功能：传送流断开的过滤器。
参数：data--传入的附加参数；
      filter_id--输入参数，与传送流断开的过滤器的ID。
返回：无。

B.4.5.3 DTVAL_onTableUpdate回调

原型：typedef void (*DTVAL_onTableUpdate)(int tunerid,int capturehandle,int version,int networkid,int orgnetid,int bouquetid,int tsid,int serviceid);
功能：PSI/SI表格更新回调函数。
参数：tunerid--Tuner标识；
      capturehandle--过滤器句柄；
      version--更新后的版本号；
      networkid--网络标识符；
      orgnetid--原始网络标识符；

bouquetid--业务群标识符；
tsid--传输流标识符；
serviceid--业务标识符。
返回：无。

B.4.6 接口定义

B.4.6.1 DTVAL_getUnusedFilterNumber接口

原型：int DTVAL_getUnusedFilterNumber();
功能：获得系统中当前未使用的段过滤器数量。
参数：无。
返回：int，系统中当前未使用的段过滤器数量。

B.4.6.2 DTVAL_requestFilter接口

原型：int DTVAL_requestFilter();
功能：申请使用一个段过滤器。
参数：无。
返回：int，系统分配的段过滤器ID。

B.4.6.3 DTVAL_releaseFilter接口

原型：int DTVAL_releaseFilter(int filter_id);
功能：释放占用的段过滤器。
参数：filter_id--输入参数，段过滤器的ID，若输入的段过滤器的ID无效，则返回ERROR_FILTER_INVALID_FILTER_ID。
返回：int，返回错误码。

B.4.6.4 DTVAL_attachStream接口

原型：int DTVAL_attachStream(int filter_id, int tuner_id, int transport_stream_id, DTVAL_reportFilterForceDisconnected *cb, void* data);
功能：将段过滤器与传送流绑定。
参数：filter_id--输入参数，段过滤器的ID，若输入的段过滤器的ID无效，则返回ERROR_FILTER_INVALID_FILTER_ID；若与段过滤器绑定的Tuner的当前传送流与参数中指定的传送流不同，则返回ERROR_FILTER_INVALID_TRANSPORT_STREAM。若绑定成功，并且段过滤器已经开始过滤，则开始上报过滤到的段数据。
tuner_id--输入参数，绑定的Tuner的ID；
transport_stream_id--输入参数，绑定的传送流的ID；
cb--回调函数，当正常过滤中发生Tuner被调谐到其他的频点或者传送流事件时调用；
data--传入回调函数的附加数据。
返回：int，返回错误码。

B.4.6.5 DTVAL_detachStream接口

原型：int DTVAL_detachStream(int filterID);

功能：将段过滤器与其绑定的传送流断开，并停止发送过滤数据。该操作不改变过滤器的开始/停止状态。

参数：filter_id--输入参数，段过滤器的ID，若输入的段过滤器的ID无效，则返回ERROR_FILTER_INVALID_FILTER_ID。

返回：int，返回错误码。

B.4.6.6 DTVAL_startFiltering接口

原型：int DTVAL_startFiltering(int filter_id, int pid, int table_id, int offset, char* pos_filter_def, char* pos_filter_mask, int pos_length, char* neg_filter_def, char* neg_filter_mask, int neg_length, DTVAL_reportFilteredSection cb, void* data);

功能：开始过滤，若本过滤器当前未与传送流绑定，则不上报过滤数据。

过滤条件：((pos_filter_def[i] & pos_filter_mask[i]) == (section_data[offset + i] & pos_filter_mask[i])) && ((neg_filter_def[i]) != (section_data[offset + i] & neg_filter_mask[i]))。

当过滤到段数据之后，通过DTVAL_reportFilteredSection回调上报。

参数：filter_id--输入参数，段过滤器的ID，若输入的段过滤器的ID无效，则返回ERROR_FILTER_INVALID_FILTER_ID；

pid--输入参数，过滤的Section的PID，若指定的pid无效则返回ERROR_FILTER_INVALID_PID；

table_id--输入参数，过滤的Section的表ID，若该参数为-1，则表示不设置表ID；

offset--输入参数，Section匹配条件，取值大于等于0；

pos_filter_def--输入参数，Section匹配条件，若为NULL，则表示不设置该条件；

pos_filter_mask--输入参数，Section匹配条件，若为NULL，则表示不设置该条件，本参数需与pos_filter_def同时使用；

pos_length--输入参数，Section匹配条件，pos_filter_def和pos_filter_mask的长度，取值大于等于0；

neg_filter_def--输入参数，Section匹配条件，若为NULL，则表示不设置该条件；

neg_filter_mask--输入参数，Section匹配条件，若为NULL，则表示不设置该条件，本参数需与neg_filter_def同时使用；

neg_length--输入参数，Section匹配条件，neg_filter_def和neg_filter_mask的长度，取值大于等于0；

cb--回调函数，当过滤到正确的段数据时调用。

data--传入回调函数的附加数据。

返回：int，返回错误码。

B.4.6.7 DTVAL_stopFiltering接口

原型：int DTVAL_stopFiltering(int filterID);

功能：停止过滤。该操作不改变过滤器与传送流的绑定状态。

参数：filterID--输入参数，段过滤器的ID，若输入的段过滤器的ID无效，则返回ERROR_FILTER_INVALID_FILTER_ID。

返回：int，返回错误码。

B.4.6.8 DTVAL_startTableMonitor接口

原型：int DTVAL_startTableMonitor(int tunerID, SI_TABLE_TYPE_E type,NGB_SERVICELOCATOR params, DTVAL_onTableUpdate cb);
功能：启动表格更新的监控，监控内容包括BAT、NIT、PAT、PMT、SDT、PF/Schedule事件，并设置回调函数。
参数：tunerID--Tuner标识；
type--表格类型；
params--业务参数，仅当type指示为PMT表时需要此参数查找PMT PID；
cb--回调函数，当监测到PSI/SI信息表版本变化调用。
返回：int，监听器标识。

B.4.6.9 DTVAL_stopTableMonitor接口

原型：void DTVAL_stopTableMonitor(int tuner, int monitorID);
功能：停止对表格更新事件的监控；
参数：tuner—Tuner标识；
monitorID--监听器标识。
返回：无。

B.4.6.10 DTVAL_getSIInfo接口

原型：int DTVAL_getSIInfo(int tunerID, DVBInfo_match_t *match, DVBInfo_sortorder_t *sortorder, void ** siInfo, int *num);
功能：获取SI信息。
参数：tunerID--输入参数，所属的TunerID，也就是数据库ID；
match--输入参数，匹配过滤所需的信息；
sortorder--输入参数，排序的依据和顺序；
siInfo--输出参数，获取到的siinfo；
num--输出参数，获取到的si信息数量。
返回：int，成功返回0，失败返回-1。

B.4.6.11 DTVAL_releaseSIInfo接口

原型：int DTVAL_releaseSIInfo(int tunerID, void *siInfo, int num);
功能：释放SI信息。
参数：tunerID--输入参数，所属的TunerID，也就是数据库ID；
siInfo--输出参数，通过DTVAL_getSIInfo获取到的siinfo；
num--输出参数，si信息数量。
返回：int，成功返回0，失败返回-1。

B.4.6.12 DTVAL_retrieveActualNetwork接口

原型：int DTVAL_retrieveActualNetwork(int tunerID,DVBNetwork_t* network);
功能：从现行NIT表中获取当前网络信息。
参数：tunerID--Tuner标识；
network--网络信息。
返回：int，成功返回0，失败返回-1。

B.4.6.13 DTVAL_retrieveActualTransportStreams接口

原型：DVBTS_t* DTVAL_retrieveActualTransportStreams(int tunerID, int* num);
功能：从现行NIT表中获取当前网络的所有传送流信息。
参数：tunerID--Tuner标识；
num--流个数。
返回：DVBTS_t*，传输流信息链表头指针。

B.4.6.14 DTVAL_releaseDVBService接口

原型：void DTVAL_releaseDVBService(DVBService_t* server);
功能：释放DVB业务结构体。
参数：server--DVB业务结构体指针。
返回：无。

B.4.6.15 DTVAL_releaseDVBTS接口

原型：void DTVAL_releaseDVBTS(DVBTS_t* TS);
功能：释放DVB传输流结构体。
参数：TS--DVB传输流结构体指针。
返回：无。

B.4.6.16 DTVAL_retrieveActualServices接口

原型：DVBService_t* DTVAL_retrieveActualServices(int tunerID, int* num);
功能：从当前传输流中获取所有业务信息。
参数：tunerID--Tuner标识。
num—业务数量
返回：DVBService_t*，业务信息链表头指针。

B.4.6.17 DTVAL_retrievePMTService接口

原型：int DTVAL_retrievePMTService(int tunerID, NGB_SERVICELOCATOR params, DVBService_t* server);
功能：获取某个业务相关的PMT描述的业务信息。
参数：tunerID--Tuner标识；
params--业务参数；
server--业务信息链表头指针。
返回：int，成功返回0，失败返回-1。

B.4.6.18 DTVAL_retrieveTimeFromTDT接口

原型：int DTVAL_retrieveTimeFromTDT(int tunerID, DVBDateTime_t* tdttime);
功能：从当前传送流承载的TDT表中获取时间信息。
参数：tunerID--Tuner标识；
tdttime--日期时间信息指针。
返回：int，成功返回0，失败返回-1。

B.4.6.19 DTVAL_retrieveTimeFromTOT接口

原型：int DTVAL_retrieveTimeFromTOT(int tunerID, DVBDateTime_t* tottime);
功能：从当前传送流承载的TOT表中获取时间信息。
参数：tunerID--Tuner标识；
tottime--日期时间信息指针。
返回：int，成功返回0，失败返回-1。

B.5 电子节目指南功能模块

B.5.1 概述

本模块定义了电子节目指南信息查询相关功能接口，见表B.5。本模块接口的实现应符合GB/T 28160—2011的相关规定。

表B.5 电子节目指南功能模块接口

接口	说明
DTVAL_EPGManager_getPresentProgram	获取指定业务的当前节目。
DTVAL_EPGManager_getPresentProgramsByContentType	根据参数中指定的节目内容分类值，在当前EPG数据库中查找符合条件的当前节目信息。
DTVAL_EPGManager_getPresentProgramsByName	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的当前节目信息。
DTVAL_EPGManager_getFollowingProgram	获取后续节目信息。
DTVAL_EPGManager_getFollowingProgramsByContentType	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的后续节目信息。
DTVAL_EPGManager_getFollowingProgramsByName	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的后续节目信息。
DTVAL_EPGManager_getProgramsByService	根据参数中指定的业务信息,获取指定业务中符合条件的节目信息。
DTVAL_EPGManager_getProgramsByDate	根据参数中指定的起始日期和结束日期,获取指定业务中符合条件的节目信息。
DTVAL_EPGManager_getProgramsByDirection	根据参数中指定的起始日期和检索方向,获取指定业务中指定个数的节目信息。
DTVAL_EPGManager_getProgramsByContentType	根据参数中指定的节目内容分类值，在当前EPG数据库中查找符合条件的节目信息。
DTVAL_EPGManager_getProgramsByName	根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的节目信息。
DTVAL_EPGManager_getReferenceEvents	获取到搜索的ReferenceEvent对象数组。
DTVAL_EPGManager_getReferencePrograms	获取指定参考业务上的参考节目。
DTVAL_ReferenceEvent_getSchedules	获取该参考事件从此时此刻起,若干天之内的所有时移事件列表，当天已经播放完毕的时移事件将被丢弃，时移事件按照起始时间进行排序。

表 B.5（续）

接口	说明
DTVAL_ReferenceEvent_getPresentSchedules	获取该参考事件所带的当前正播放的所有时移事件对象，数组中的元素按照播放起始时间进行排序。
DTVAL_ReferenceEvent_getFollowingSchedules	获取所有的时移事件信息。
DTVAL_EPGManager_getPresentProgram	获取指定业务的当前节目。
DTVAL_EPGManager_getPresentProgramsByContentType	根据参数中指定的节目内容分类值，在当前EPG数据库中查找符合条件的当前节目信息。
DTVAL_EPGManager_getAllProgramServices	返回所有的Service节目信息。
DTVAL_releaseJS_PROGRAMEVENT_PRIVATE	释放JS_PROGRAMEVENT_PRIVATE 结构体。

B.5.2 常量定义

原型:#define NGB_EPG_LANGUAGE 0x636869
描述：EPG语言编码，表示ASCII汉语。

B.5.3 事件类型

无。

B.5.4 数据结构

B.5.4.1 时间信息

```
原型：typedef struct {
    int second;
    int minute;
    int hour;
    int day;
    int month;
    int year;
} DVBDateTime_t;
```

描述：时间信息。
成员：
second：秒(0 - 59)；
minute：分(0 - 59)；
hour：时(0 - 23)；
day：日(1 - 31)；
month：月(0 - 11)；
year：年(past 1900)。

B.5.4.2 节目事件信息

```
原型：typedef struct {
    unsigned short network_id;
    unsigned short original_network_id;
```

```
    unsigned short transport_stream_id;
    unsigned short service_id;
    unsigned short event_id;
    char event_name[MAX_NAME_LENGTH];
    char short_event_name[MAX_NAME_LENGTH];
    char lingual[4];
    char event_desc[MAX_NAME_LENGTH];
    unsigned char running_status;
    DVBDateTime_t start_time;
    unsigned int duration;
    unsigned char content_nibble;
    unsigned char user_nibble;
    unsigned char min_age;
    unsigned char free_ca_mode;
    unsigned char ca_lock_mode;
    DVBDescriptor_t* descriptors;
} DVBEvent_t;
```

描述：节目事件信息。

成员：

network_id：网络ID；

original_network_id：原始网络ID；

transport_stream_id：传输流ID；

service_id：业务ID；

event_id：事件ID；

event_name[MAX_NAME_LENGTH]：事件名称；

short_event_name[MAX_NAME_LENGTH]：事件名称；

lingual[4]：语言；

event_desc[MAX_NAME_LENGTH]：短事件描述；

running_status：运行状态；

start_time：开始事件；

duration：时长（单位：秒）；

content_nibble：内容nibble；

user_nibble：用户nibble；

min_age：最小观看年龄；

free_ca_mode：是否清流；

ca_lock_mode：ca 锁定模式；

descriptors：描述符。

B.5.4.3 EPG监控信息

```
原型：typedef struct {
    int m_type;
    int m_mask;
```

```
    int m_timeout;
    int m_maxnum;
    int m_startDay;
    int m_searchDay;
    int m_tsNum;
    DVBTransport_stream_match_t m_transportStreams[64];
    DVBService_match_t m_service;
} DVBEPGInfo_match_t;
```

描述：EPG监控信息。

成员：

m_type：类型；

m_mask：mask；

m_timeout：超时时长（单位：秒）；

m_maxnum：最大个数；

m_startDay：起始时间；

m_searchDay：要搜索的天数；

m_tsNum：TS个数；

m_transportStreams[64]：ts信息；

m_service：节目信息。

B.5.4.4 预定信息

```
原型：typedef struct _ORDER_ {
    JS_CHANNEL_PRIVATE *pChannel;
    void *eventObj;
    int orderID;
    unsigned int deleteFlag;
    unsigned short type;
    unsigned short status;
} JS_ORDER_PRIVATE;
```

描述：预定信息。

成员：

pChannel：节目信息；

eventObj：事件信息；

orderID：预定ID；

deleteFlag：是否删除Flag；

type：类型；

status：状态。

B.5.4.5 EPG信息结构

```
原型：typedef struct {
    JS_CHANNEL_PRIVATE channelObj;
    DVBEvent_t eventObj;
```

```
    int isBooked;
} JS_PROGRAMEVENT_PRIVATE;
```

描述：EPG信息。

成员：

channelObj：节目信息；

eventObj：事件信息；

isBooked：是否预定。

B.5.4.6 时移信息结构

```
原型：typedef struct {
    JS_CHANNEL_PRIVATE channelObj;
    JS_CHANNEL_PRIVATE refChannelObj;
    DVBEvent_t eventObj;
    DVBEvent_t refEventObj;
    int status;
    int orderIndex;
} JS_TIMESHIFTEVENT_PRIVATE;
```

描述：时移信息。

成员：

channelObj：节目信息；

refChannelObj：参考节目信息；

eventObj：事件信息；

refEventObj：参考事件信息；

status：状态；

orderIndex：order索引。

B.5.5 回调函数定义

无。

B.5.6 接口定义

B.5.6.1 DTVAL_EPGManager_getPresentProgram接口

```
原型：int DTVAL_EPGManager_getPresentProgram(int tunerID, NGB_SERVICELOCATOR
      serviceLocator, JS_PROGRAMEVENT_PRIVATE *programevent);
```

功能：同步方法，获取指定业务的当前节目。

参数：tunerID--Tuner标识；

serviceLocator—业务描述符；

programevent—节目内容分类值。

返回：int，0表示成功，其它为失败。

B.5.6.2 DTVAL_EPGManager_getPresentProgramsByContentType接口

原型：int DTVAL_EPGManager_getPresentProgramsByContentType(int tunerID,int contentType,
JS_PROGRAMEVENT_PRIVATE **programevent,int *objnumber);
功能：同步方法，根据参数中指定的节目内容分类值，在当前EPG数据库中查找符合条件的当前节目信息。
参数：tunerID--Tuner标识；
contentType--int型，表示节目内容分类类型；
programevent--_PROGRAMEVENT_PRIVATE数据结构数组，表示节目内容分类值；
objnumber--int型，JS_PROGRAMEVENT_PRIVATE数据结构的个数；
返回：int，0表示成功，其它为失败。

B.5.6.3 DTVAL_EPGManager_getPresentProgramsByName接口

原型：int DTVAL_EPGManager_getPresentProgramsByName(int tunerID, char *str,
JS_PROGRAMEVENT_PRIVATE **programevent, int* Objnumber);
功能：同步方法，根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的当前节目信息。
参数：tunerID--Tuner标识；
str--sring型，表示搜索关键字；
programevent--_PROGRAMEVENT_PRIVATE数据结构数组,表示节目名称。
objnumber--int型，JS_PROGRAMEVENT_PRIVATE数据结构的个数；
返回：int，0表示成功，其它为失败。

B.5.6.4 DTVAL_EPGManager_getFollowingProgram接口

原型：int DTVAL_EPGManager_getFollowingProgram(int tunerID,
NGB_SERVICELOCATOR serviceLocator, JS_PROGRAMEVENT_PRIVATE *programevent);
功能：同步方法，获取后续节目信息。
参数：tunerID--Tuner标识；
serviceLocator--表示广播业务定位符；
programevent--表示节目信息。
返回：int，0表示成功，其它为失败。

B.5.6.5 DTVAL_EPGManager_getFollowingProgramsByContentType接口

原型：int DTVAL_EPGManager_getFollowingProgramsByContentType(int tunerID,int contentType,
JS_PROGRAMEVENT_PRIVATE **programevent, int* objnumber);
功能：同步方法，根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的后续节目信息。
参数：tunerID--Turner标识；
contentType--表示节目内容分类类型；
programevent--JS_PROGRAMEVENT_PRIVATE数据结构数组，表示节目名称；
objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。
返回：int，0表示成功，其它为失败。

B.5.6.6 DTVAL_EPGManager_getFollowingProgramsByName接口

原型：int DTVAL_EPGManager_getFollowingProgramsByName(int tunerID,char *str,
JS_PROGRAMEVENT_PRIVATE **programevent, int *objnumber);

功能：同步方法，根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的后续节目信息。
参数：tunerID--Tuner标识；
str--字符串型，表示搜索关键字；
programevent--JS_PROGRAMEVENT_PRIVATE数据结构数组,表示节目名称;
objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数；
返回：int，0表示成功，其它为失败。

B.5.6.7 DTVAL_EPGManager_getProgramsByService接口

原型：int DTVAL_EPGManager_getProgramsByService(int tunerID, NGB_SERVICELOCATOR serviceLocator, JS_PROGRAMEVENT_PRIVATE **programevent, int *objnumber);
功能：根据参数中指定的业务信息，获取指定业务中符合条件的节目信息。
参数：tunerID--Tuner标识；
serviceLocator--NGB_SERVICELOCATOR 数据结构，表示广播业务定位符；
programevent--JS_PROGRAMEVENT_PRIVATE数据结构数组；
objnumber--int型JS_PROGRAMEVENT_PRIVATE数据结构的个数。
返回：int，0表示成功，其它为失败。

B.5.6.8 DTVAL_EPGManager_getProgramsByDate接口

原型：int DTVAL_EPGManager_getProgramsByDate(int tunerID,
NGB_SERVICELOCATOR serviceLocator, DVBDateTime_t* beginDate,
DVBDateTime_t *endDate, JS_PROGRAMEVENT_PRIVATE **programevent,
int *objnumber);
功能：同步方法，根据参数中指定的起始日期和结束日期，获取指定业务中符合条件的节目信息。
参数：tunerID--Tuner标识；
serviceLocator--NGB_SERVICELOCATOR 数据结构表示广播业务定位符；
beginDate--JS_DATE_PRIVATE数据结构，表示起始日期；
endDate--JS_DATE_PRIVATE数据结构，表示结束日期；
programevent--表示节目信息；
objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。
返回：int，0表示成功，其它为失败。

B.5.6.9 DTVAL_EPGManager_getProgramsByDirection接口

原型：int DTVAL_EPGManager_getProgramsByDirection(int tunerID,
NGB_SERVICELOCATOR serviceLocator, DVBDateTime_t *beginDate,
int count, int isForward, JS_PROGRAMEVENT_PRIVATE *programevent,
int *objnimber);
功能：同步方法，根据参数中指定的起始日期和检索方向，获取指定业务中指定个数的节目信息。
参数：tunerID--Tuner标识；
serviceLocator--表示广播业务定位符；
beginDate--表示起始日期；
count--表示待获取的节目信息个数；
isForword--0表示向后搜索；1表示向前搜索；

programevent--表示节目信息；
objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。
返回：int，0表示成功，其它为失败。

B.5.6.10 DTVAL_EPGManager_getProgramsByContentType接口

原型：int DTVAL_EPGManager_getProgramsByContentType(int tunerID, int contentType,
JS_PROGRAMEVENT_PRIVATE **programevent, int *objnumber);
功能：同步方法，根据参数中指定的节目内容分类值，在当前EPG数据库中查找符合条件的节目信息。
参数：tunerID--Tuner标识；
contentType--int型，节目内容分类类型；
programevent--表示节目信息；
objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。
返回：int，0表示成功，其它为失败。

B.5.6.11 DTVAL_EPGManager_getProgramsByName接口

原型：int DTVAL_EPGManager_getProgramsByName(int tunerID, char *str,
JS_PROGRAMEVENT_PRIVATE **programevent, int *objnumber);
功能：同步方法，根据参数中指定的节目名称，在当前EPG数据库中查找符合条件的节目信息。
参数：tunerID--Tuner标识；
str--输入参数，字符串型，表示搜索关键字；
programevent--表示节目信息；
objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。
返回：int，0表示成功，其它为失败。

B.5.6.12 DTVAL_EPGManager_getReferenceEvents接口

原型：int DTVAL_EPGManager_getReferenceEvents(int tunerID,int sortType,int sortOrder,
JS_PROGRAMEVENT_PRIVATE **programevent, int *objnumber);
功能:获取到搜索的ReferenceEvent对象数组。
参数：tunerID--Tuner标识；
sortType--int型，表示排序依据，取值：
--1：代表根据参考事件ID排序；
--2：代表根据参考事件名称排序。
sortOrder--int型，表示排序方式，取值：
--0：表示降序排序；
--1：表示升序排序。
programevent—表示节目信息；
objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。
返回：int，0表示成功，其它为失败。

B.5.6.13 DTVAL_EPGManager_getReferencePrograms接口

原型：int DTVAL_EPGManager_getReferencePrograms(int tunerID,
NGB_SERVICELOCATOR serviceLocator, JS_PROGRAMEVENT_PRIVATE **programevent,

int *objnumber);

功能：同步方法，获取指定参考业务上的参考节目。

参数：tunerID--Tuner标识；

serviceLocator--表示广播业务定位符；

programevent—表示节目信息；

objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。

返回：int，0表示成功，其它为失败。

B.5.6.14 DTVAL_ReferenceEvent_getSchedules接口

原型：int DTVAL_ReferenceEvent_getSchedules(int tunerID,
JS_TIMESHIFTEVENT_PRIVATE **timeshiftevent, int *objnumber);

功能：获取该参考事件从此时此刻起，若干天之内的所有时移事件列表，当天已经播放完毕的时移事件将被丢弃，时移事件按照起始时间进行排序。

参数：tunerID--Tuner标识；

programevent—表示节目信息；

objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。

返回：int，0表示成功，其它为失败。

B.5.6.15 DTVAL_ReferenceEvent_getPresentSchedules接口

原型：int DTVAL_ReferenceEvent_getPresentSchedules(int tunerID,
JS_TIMESHIFTEVENT_PRIVATE **timeshiftevent, int *objnumber);

功能：获取该参考事件所带的当前正播放的所有时移事件对象，数组中的元素按照播放起始时间进行排序。

参数：tunerID--Tuner标识；

programevent—表示节目信息；

objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。

返回：int，0表示成功，其它为失败。

B.5.6.16 DTVAL_ReferenceEvent_getFollowingSchedules接口

原型：int DTVAL_ReferenceEvent_getFollowingSchedules(int tunerID,
JS_TIMESHIFTEVENT_PRIVATE **timeshiftevent, int *objnumber);

功能：异步方法，获取所有的时移事件信息。

参数：tunerID--Tuner标识；

programevent—表示节目信息；

objnumber--JS_PROGRAMEVENT_PRIVATE数据结构的个数。

返回：int，0表示成功，其它为失败。

B.5.6.17 DTVAL_EPGManager_getAllProgramServices接口

原型：void DTVAL_EPGManager_getAllProgramServices(int tunerID, DVBService_t **services, int *objnumber);

功能：同步方法，返回所有的Service节目信息。

参数：tunerID--Tuner标识；

services—所有节目信息组成的DVBService_t的链表头指针；
objnumber-- DVBService_t数据结构的个数。

返回：无。

B.5.6.18 DTVAL_releaseJS_PROGRAMEVENT_PRIVATE接口

原型：void DTVAL_releaseJS_PROGRAMEVENT_PRIVATE(JS_PROGRAMEVENT_PRIVATE* programevent);
功能：释放JS_PROGRAMEVENT_PRIVATE结构体。
参数：programevent--JS_PROGRAMEVENT_PRIVATE结构体指针。
返回：无。

B.6 节目信息管理功能模块

B.6.1 概述

本模块定义了节目信息查询和管理相关功能接口，见表B.6。

表B.6 节目信息管理功能模块接口

接口	说明
DTVAL_ChannelManager_getChannelByChannelID	根据频道ID获取频道对象。
DTVAL_ChannelManager_getChannelByLogicalID	根据逻辑频道号获取频道对象。
DTVAL_ChannelManager_getChannelByServiceID	根据业务ID获取频道对象。
DTVAL_ChannelManager_getLastChannel	获取前一个打开的指定业务类型的频道。
DTVAL_ChannelManager_getShutDownChannel	获取指定类型的关机频道。
DTVAL_ChannelManager_delChannel	从频道列表中删除指定的频道。
DTVAL_ChannelManager_deleteAllChannels	从频道列表中删除所有的频道。
DTVAL_ChannelManager_deleteAllFavorites	从频道列表中删除所有的喜爱频道。
DTVAL_ChannelManager_deleteAllDelMarkedChannels	从频道列表中删除所有带删除标记的频道。
DTVAL_ChannelManager_resetProperties	将用户所有设置为喜爱、锁定、隐藏等标记的频道全部重置，所有的频道全部更改为非喜爱、非锁定、非隐藏。
DTVAL_ChannelManager_swapChannel	交换频道对象obj1和频道对象obj2在频道列表中的位置。
DTVAL_ChannelManager_sortChannels	按照指定的方式进行频道排序。
DTVAL_ChannelManager_filterChannels	在当前频道列表中过滤出指定条件的新的频道列表。
DTVAL_ChannelManager_saveChannels	将RAM中的频道列表数据保存到NVM中。
DTVAL_ChannelManager_restoreChannels	将NVM中的频道列表数据恢复到RAM中。
DTVAL_ChannelManager_getServiceByChannel	获取当前频道对象对应的DvbService对象。
DTVAL_ChannelManager_updateChannel	将应用设置的channel属性同步到数据库中。
DTVAL_releaseJS_CHANNEL_PRIVATE	释放JS_CHANNEL_PRIVATE 结构体。
DTVAL_ChannelManager_getTunerIDOfLastChannel	获取最后保存节目的TunerID。
DTVAL_SetCarriesList	存储频点列表。
DTVAL_SetServiceList	存储节目列表。
DTVAL_UpdateChannelData	更新节目信息。
DTVAL_ChannelManager_getChannelByServiceType	根据业务类型获取频道对象。

B.6.2 常量定义

无。

B.6.3 事件类型

无。

B.6.4 数据结构

B.6.4.1 节目信息基础数据

```
原型：typedef struct {
    int network_id;
    int onet_id;
    int ts_id;
    int service_id;
    int service_type;
    int logic_channel;
    int channel_id;
    int deltVolume;
    int isHided;
    int isDeleted;
    int isFavorite;
    int isLocked;
    int supportPlayback;
} JS_CHANNEL_PRIVATE;
```

描述：节目信息基础数据。

成员：

network_id：网络ID；

onet_id：原始网络ID；

ts_id：传输流ID；

service_id：节目ID；

service_type：节目类型；

logic_channel：逻辑频道号；

channel_id：频道ID；

deltVolume：音量补偿；

isHided：是否隐藏；

isDeleted：是否删除；

isFavorite：是否喜爱；

isLocked：是否加锁；

supportPlayback：是否支持回放。

B.6.5 回调函数定义

无。

B.6.6 接口定义

B.6.6.1 DTVAL_ChannelManager_getChannelByChannelID接口

原型：int DTVAL_ChannelManager_getChannelByChannelID(unsigned short channelId,
JS_CHANNEL_PRIVATE *channel);
功能：根据频道ID获取频道对象。
参数：channelId--频道ID，大于0的整数。
channel--获取到的channel对象。
返回：int，0表示成功，2表示错误。

B.6.6.2 DTVAL_ChannelManager_getChannelByLogicalID接口

原型：int DTVAL_ChannelManager_getChannelByLogicalID(unsigned short logicId,
JS_CHANNEL_PRIVATE *channel);
功能：根据逻辑频道号获取频道对象。
参数：logicId--逻辑频道号，大于0的整数。
channel--获取到的channel对象。
返回：int，0表示成功，2表示错误。

B.6.6.3 DTVAL_ChannelManager_getChannelByServiceID接口

原型：int DTVAL_ChannelManager_getChannelByServiceID(unsigned short serviceId,
JS_CHANNEL_PRIVATE *channel);
功能：根据业务ID获取频道对象。
参数：serviceId--业务ID，大于0的整数。
channel--获取到的channel对象。
返回：int，0表示成功，2表示错误。

B.6.6.4 DTVAL_ChannelManager_getLastChannel接口

原型：int DTVAL_ChannelManager_getLastChannel(unsigned short serviceType,
JS_CHANNEL_PRIVATE *channel);
功能：获取前一个打开的指定业务类型的频道。
参数：serviceType--指定的业务类型(0xffff 表示不关心类型)。
channel--获取到的channel对象。
返回：int，0表示成功，2表示错误。

B.6.6.5 DTVAL_ChannelManager_getShutDownChannel接口

原型：int DTVAL_ChannelManager_getShutDownChannel(unsigned short serviceType,
JS_CHANNEL_PRIVATE *channel);
功能：获取指定类型的关机频道。
参数：serviceType--指定的业务类型。
channel--获取到的channel对象。
返回：int，0表示成功，2表示错误。

B.6.6.6 DTVAL_ChannelManager_delChannel接口

原型：int DTVAL_ChannelManager_delChannel(JS_CHANNEL_PRIVATE *channel);
功能：从频道列表中删除指定的频道。
参数：channel--channel对象。
返回：int，0表示成功，2表示错误。

B.6.6.7 DTVAL_ChannelManager_deleteAllChannels接口

原型：int DTVAL_ChannelManager_deleteAllChannels(void);
功能：从频道列表中删除所有的频道。
参数：无。
返回：int，0表示成功，2表示错误。

B.6.6.8 DTVAL_ChannelManager_deleteAllFavorites接口

原型：int DTVAL_ChannelManager_deleteAllFavorites(void);
功能：从频道列表中删除所有的喜爱频道。
参数：无。
返回：int，0表示成功，2表示错误。

B.6.6.9 DTVAL_ChannelManager_deleteAllDelMarkedChannels接口

原型：int DTVAL_ChannelManager_deleteAllDelMarkedChannels(void);
功能：从频道列表中删除所有带删除标记的频道。
参数：无。
返回：int，0表示成功，2表示错误。

B.6.6.10 DTVAL_ChannelManager_resetProperties接口

原型：int DTVAL_ChannelManager_resetProperties(void);
功能：将用户所有设置为喜爱、锁定、隐藏等标记的频道全部重置，所有的频道全部更改为非喜爱、非锁定、非隐藏。
参数：无。
返回：int，0表示成功，2表示错误。

B.6.6.11 DTVAL_ChannelManager_swapChannel接口

原型：int DTVAL_ChannelManager_swapChannel(JS_CHANNEL_PRIVATE *channel1,
JS_CHANNEL_PRIVATE *channel2);
功能：交换频道对象obj1和频道对象obj2在频道列表中的位置
参数：channel1--频道1(obj1)；
channel2--频道2(obj2)。
返回：int，0表示成功，2表示错误。

B.6.6.12 DTVAL_ChannelManager_sortChannels接口

原型：int DTVAL_ChannelManager_sortChannels(int *sortTypeArray, int sortTypeNum,
int *sortModeArray, int sortModeNum);
功能：按照指定的方式进行频道排序。

参数：sortTypeArray--排序依据；
sortModeArray--排序方式。
返回：int，0表示成功，2表示错误。

B.6.6.13 DTVAL_ChannelManager_filterChannels接口

原型：int DTVAL_ChannelManager_filterChannels(int filterTypeArray[], int valueArray[], int num, JS_CHANNEL_PRIVATE **listChannels, int *channelNums);
功能：在当前频道列表中过滤出指定条件的新的频道列表。
参数：filterTypeArray--排序依据；
valueArray--排序方式。
listChannels--排序后的频道数组；
channelNumbs--频道数组长度。
返回：int，0表示成功，2表示错误。

B.6.6.14 DTVAL_ChannelManager_saveChannels接口

原型：int DTVAL_ChannelManager_saveChannels(void);
功能：异步方法，将RAM中的频道列表数据保存到NVM中。
参数：无。
返回：int，0表示成功，2表示错误。

B.6.6.15 DTVAL_ChannelManager_restoreChannels接口

原型：int DTVAL_ChannelManager_restoreChannels(void);
功能：异步方法，将NVM中的频道列表数据恢复到RAM中。
参数：无。
返回：int，0表示成功，2表示错误。

B.6.6.16 DTVAL_ChannelManager_getServiceByChannel接口

原型：int DTVAL_ChannelManager_getServiceByChannel(JS_CHANNEL_PRIVATE *channel, DVBService_t* server);
功能： 获取当前频道对象对应的DvbService对象。
参数：channel--当前频道对象；
server--业务信息链表头指针。
返回：int，0表示成功，2表示错误。

B.6.6.17 DTVAL_ChannelManager_updateChannel接口

原型：int DTVAL_ChannelManager_updateChannel(JS_CHANNEL_PRIVATE *channel);
功能：将应用设置的channel属性同步到数据库中。
参数：channel--频道对象。
返回：int，0表示成功，2表示错误。

B.6.6.18 DTVAL_releaseJS_CHANNEL_PRIVATE接口

原型：void DTVAL_releaseJS_CHANNEL_PRIVATE(JS_CHANNEL_PRIVATE* listChannels);

功能：释放JS_CHANNEL_PRIVATE 结构体。
参数：listChannels—JS_CHANNEL_PRIVATE结构体指针。
返回：int，0表示成功，-1表示错误。

B.6.6.19 DTVAL_ChannelManager_getTunerIDOfLastChannel接口

原型：int DTVAL_ChannelManager_getTunerIDOfLastChannel(int *rpi_tunerID);
功能：获取最后保存节目的tunerid。
参数：rpi_tunerID—int指针，用于存放tunerid。
返回：int，0表示成功，-1表示错误。

B.6.6.20 DTVAL_SetCarriesList接口

原型：int DTVAL_SetCarriesList(DVBCarrierInfo_t* delivery, int nums);
功能：存储频点列表。
参数：delivery：频点信息；
nums：频点个数。
返回：int，0表示成功，-1表示错误。

B.6.6.21 DTVAL_SetServiceList接口

原型：int DTVAL_SetServiceList(int tunerID,DVBService_t* service,int nums);
功能：存储节目列表。
参数：tunerID--数据库ID；
service--节目信息；
nums--节目个数。
返回：int，0表示成功，-1表示错误。

B.6.6.22 DTVAL_UpdateChannelData接口

原型：int DTVAL_UpdateChannelData(int tunerID,DVBService_t service);
功能：更新节目信息。
参数：tunerID--数据库ID；
service--节目信息。
返回：int，0表示成功，-1表示错误。

B.6.6.23 DTVAL_ChannelManager_getChannelByServiceType接口

原型：int DTVAL_ChannelManager_getChannelByServiceType(int tunerID,
unsigned short ServiceType, JS_CHANNEL_PRIVATE *channel, int *objnumber);
功能：根据业务类型获取频道对象;
参数：tunerID--数据库ID;
ServiceType--业务类型(大于0的整数);
channel--节目数据（传入必须为有效指针）；
objnumber--channel最大可用于存储的节目个数。
返回：int，0表示成功，-1表示错误。

附 录 C
（规范性附录）
媒体引擎组件

C.1 概述

本附录定义了媒体引擎组件对外提供的接口，包括媒体播放功能模块对外接口,见表 C.1。

表 C.1 媒体引擎组件功能模块

模块	说明
媒体播放功能模块	定义了媒体播放接口的常量、事件类型、数据结构和接口。

C.2 媒体播放功能模块

C.2.1 概述

本模块定义了媒体播放接口的常量、事件类型、数据结构和接口，见表C.2。

表C.2 媒体播放功能模块接口

接口	说明
setDataSource	以 path 和 Hash Key 对的方式设置数据源。
setDataSource	以文件句柄方式设置数据源。
setVideoSurfaceTexture	设置视频显示画幕。
setListener	设置媒体播放监听器。
prepare	同步的方式让播放器准备，直到播放器准备好才返回。
prepareAsync	异步的方式让播放器准备，调用后该接口立刻返回。
start	从 pause 或者 stop 状态启动播放。
stop	停止播放。
pause	暂停播放。
isPlaying	获取播放器状态是否处于播放中。
getVideoWidth	获取视频播放窗口宽度。
getVideoHeight	获取视频播放窗口高度。
setVideoArea	设置视频播放窗口区域。
getVideoArea	获取视频播放窗口区域。
seekTo	让播放器在指定的位置播放。
getCurrentPosition	获取当前播放位置。
getDuration	获取播放源的可播放长度。
reset	重置播放器。
setPace	设置播放器速率。
getPace	获取播放器速率。
setStopMode	设置媒体播放暂停效果。
getStopMode	获取媒体播放暂停效果。

表 C.2（续）

接口	说明
setClip	设置视频源图像的剪切区域。
getClip	获取视频源图像的剪切区域。
getStartTime	获取时移（或者回看）节目的起始时间。
selectAudioStream	选择当前播放器的音频流。
setLooping	设置播放器是否循环播放。
isLooping	获取当前状态是否为循环播放。
setDisplayMode	设置播放器显示模式。
setDisplayRatio	设置播放时视频的宽高比率。
setMetadataFilter	设置媒体元数据过滤器。
getMetadata	获取媒体元数据。
setVideoDisplay	设置视频输出。
getVideoDisplay	获取视频输出状态。

C.2.2 常量定义

C.2.2.1 媒体播放错误码

```
原型：typedef enum __tagSmePlayerError {
    E_SME_PLAYER_ERROR_NONE = 0,
    E_SME_PLAYER_ERROR_UNKOWN = 10000,
    E_SME_PLAYER_ERROR_TIME_OUT = 10001,
    E_SME_PLAYER_ERROR_UNSUPPORT_FORMAT = 10002,
    E_SME_PLAYER_ERROR_UNSUPPORT_AUDIO_CODEC = 10003,
    E_SME_PLAYER_ERROR_UNSUPPORT_VIDEO_CODEC = 10004,
    E_SME_PLAYER_ERROR_UNSUPPORT_DISPLAY_MODE = 10005,
    E_SME_PLAYER_ERROR_UNSUPPORT_DISPLAY_ANGLE = 10006,
    E_SME_PLAYER_ERROR_UNSUPPORT_PLAY_MODE = 10007,
    E_SME_PLAYER_ERROR_UNSUPPORT_OP = 10008,
    E_SME_PLAYER_ERROR_STATE = 10009,
    E_SME_PLAYER_ERROR_NOMEM = 10010,
    E_SME_PLAYER_ERROR_INVALID_ARGS =10011,
    E_SME_PLAYER_ERROR_ASYNC = 10012,
    E_SME_PLAYER_ERROR_UNSUPPORT_SEEK = 10013,
    E_SME_PLAYER_ERROR_UNSUPPORT_TEXT_CODEC = 10014,
    E_SME_PLAYER_ERROR_SUBTILE_NOT_FOUND = 10015,
    E_SME_PLAYER_ERROR_SUBTILE_ACCESS = 10016,
    E_SME_PLAYER_ERROR_FITIN_PARAMETER = 10017,
    E_SME_PLAYER_ERROR_INVAILD_RECT = 10018,
    E_SME_PLAYER_ERROR_UNSUPPORT_LOOP_PLAYBACK = 10019,
    E_SME_PLAYER_ERROR_UNSUPPORT_DISPLAY_RATIO = 10020,
```

```
    E_SME_PLAYER_ERROR_INVALID_DRMID = 10021,
    E_SME_PLAYER_ERROR_STREAM_FAILED = 10022,
} E_SME_PLAYER_ERROR;
```

描述：媒体播放错误码定义。

成员：

E_SME_PLAYER_ERROR_NONE：无错误；

E_SME_PLAYER_ERROR_UNKOWN：未知错误；

E_SME_PLAYER_ERROR_TIME_OUT：操作超时；

E_SME_PLAYER_ERROR_UNSUPPORT_FORMAT：不支持格式；

E_SME_PLAYER_ERROR_UNSUPPORT_AUDIO_CODEC：不支持音频格式；

E_SME_PLAYER_ERROR_UNSUPPORT_VIDEO_CODEC：不支持视频格式；

E_SME_PLAYER_ERROR_UNSUPPORT_DISPLAY_MODE：不支持显示模式；

E_SME_PLAYER_ERROR_UNSUPPORT_DISPLAY_ANGLE：不支持显示；

E_SME_PLAYER_ERROR_UNSUPPORT_PLAY_MODE：不支持播放模式；

E_SME_PLAYER_ERROR_UNSUPPORT_OP：不支持操作；

E_SME_PLAYER_ERROR_STATE：错误播放状态；

E_SME_PLAYER_ERROR_NOMEM：无内存；

E_SME_PLAYER_ERROR_INVALID_ARGS：非法参数；

E_SME_PLAYER_ERROR_ASYNC：异步操作失败；

E_SME_PLAYER_ERROR_UNSUPPORT_SEEK：不支持SEEK操作；

E_SME_PLAYER_ERROR_UNSUPPORT_TEXT_CODEC：不支持文字格式；

E_SME_PLAYER_ERROR_SUBTILE_NOT_FOUND：字幕未发现；

E_SME_PLAYER_ERROR_SUBTILE_ACCESS：获取字幕失败；

E_SME_PLAYER_ERROR_FITIN_PARAMETER：错误的FITIN参数；

E_SME_PLAYER_ERROR_INVAILD_RECT：非法的窗口参数；

E_SME_PLAYER_ERROR_UNSUPPORT_LOOP_PLAYBACK：不支持循环播放；

E_SME_PLAYER_ERROR_UNSUPPORT_DISPLAY_RATIO：不支持的显示比例；

E_SME_PLAYER_ERROR_INVALID_DRMID：非法的DRM编码；

E_SME_PLAYER_ERROR_STREAM_FAILED：获取流失败。

C.2.2.2 媒体引擎错误码

```
原型：enum media_error_type {
    MEDIA_ERROR_UNKNOWN = 1,
    MEDIA_ERROR_SERVER_DIED = 100,
    MEDIA_ERROR_NOT_VALID_FOR_PROGRESSIVE_PLAYBACK = 200,
    TVOS_MEDIA_ERROR_START_FAILD = 1000,
    TVOS_MEDIA_ERROR_SETPACE_FAILED = 1001,
    TVOS_MEDIA_ERROR_SEEK_FAILD = 1002,
    TVOS_MEDIA_ERROR_PAUSE_FAILD = 1003,
    TVOS_MEDIA_ERROR_RESUME_FAILD = 1004,
    TVOS_MEDIA_ERROR_STOP_FAILD = 1005,
    TVOS_MEDIA_ERROR_URL_INVALID = 1006,
```

```
    TVOS_MEDIA_ERROR_RESOURCE_UNAVAILABLE = 1007,
    TVOS_MEDIA_ERROR_AUDIO_DECODE_ERROR = 1008,
    TVOS_MEDIA_ERROR_VIDEO_DECODE_ERROR = 1009,
    TVOS_MEDIA_ERROR_UNSUPPORT_VIDEO_DEC = 1010,
    TVOS_MEDIA_ERROR_UNSUPPORT_AUDIO_DEC = 1011,
    TVOS_MEDIA_ERROR_CONNECT_FAILED = 1012,
    TVOS_MEDIA_ERROR_VOD_SEARCH_FAILED = 1300,
    TVOS_MEDIA_ERROR_VOD_OUT_OF_RANGE = 1301,
};
```

描述：媒体错误类型定义。

成员：

MEDIA_ERROR_UNKNOWN：未知错误；

MEDIA_ERROR_SERVER_DIED：媒体服务死亡；

MEDIA_ERROR_NOT_VALID_FOR_PROGRESSIVE_PLAYBACK：渐进回放不合法；

TVOS_MEDIA_ERROR_START_FAILD：启播失败；

TVOS_MEDIA_ERROR_SETPACE_FAILED：设置倍速播放失败；

TVOS_MEDIA_ERROR_SEEK_FAILD： SEEK失败；

TVOS_MEDIA_ERROR_PAUSE_FAILD：暂停播放失败；

TVOS_MEDIA_ERROR_RESUME_FAILD：恢复播放失败；

TVOS_MEDIA_ERROR_STOP_FAILD：停止播放失败；

TVOS_MEDIA_ERROR_URL_INVALID：设置播放路径无效；

TVOS_MEDIA_ERROR_RESOURCE_UNAVAILABLE：播放资源不可用；

TVOS_MEDIA_ERROR_AUDIO_DECODE_ERROR：音频解码错误；

TVOS_MEDIA_ERROR_VIDEO_DECODE_ERROR：视频解码错误；

TVOS_MEDIA_ERROR_UNSUPPORT_VIDEO_DEC：不支持视频解码；

TVOS_MEDIA_ERROR_UNSUPPORT_AUDIO_DEC：不支持音频解码；

TVOS_MEDIA_ERROR_CONNECT_FAILED：连接服务器失败，建立会话失败或者服务器返回超时；

TVOS_MEDIA_ERROR_VOD_SEARCH_FAILED：搜索数据失败，用于IPQAM播放；

TVOS_MEDIA_ERROR_VOD_OUT_OF_RANGE：时间超出有效范围，用于IPQAM播放。

媒体引擎错误码随着业务丰富会逐步扩展，媒体引擎错误码扩展规则定义见表C.3。

表C.3 媒体引擎错误码扩展规则

扩展段取值	扩展业务范围
1000~1099	TVOS 公共消息扩展
1100~1199	DVB 业务扩展
1200~1299	CA 扩展
1300~1399	VOD 业务扩展
1400~1499	云游戏业务扩展
1500~1599	VideoPhone 业务扩展

C.2.2.3 媒体播放状态类型

原型：enum media_player_states {

```
    MEDIA_PLAYER_STATE_ERROR        = 0,
    MEDIA_PLAYER_IDLE               = 1 << 0,
    MEDIA_PLAYER_INITIALIZED        = 1 << 1,
    MEDIA_PLAYER_PREPARING          = 1 << 2,
    MEDIA_PLAYER_PREPARED           = 1 << 3,
    MEDIA_PLAYER_STARTED            = 1 << 4,
    MEDIA_PLAYER_PAUSED             = 1 << 5,
    MEDIA_PLAYER_STOPPED            = 1 << 6,
    MEDIA_PLAYER_PLAYBACK_COMPLETE  = 1 << 7,
    MEDIA_PLAYER_SETPACE            = 1 << 8,
    MEDIA_PLAYER_RESUME             = 1 << 9
};
```

描述：媒体播放状态类型定义。

成员：

MEDIA_PLAYER_STATE_ERROR：错误状态；

MEDIA_PLAYER_IDLE：空闲状态；

MEDIA_PLAYER_INITIALIZED：初始化完成状态；

MEDIA_PLAYER_PREPARING：准备中状态；

MEDIA_PLAYER_PREPARED：准备完成状态；

MEDIA_PLAYER_STARTED：开始播放状态；

MEDIA_PLAYER_PAUSED：暂停状态；

MEDIA_PLAYER_STOPPED：停止状态；

MEDIA_PLAYER_PLAYBACK_COMPLETE：播放完成状态；

MEDIA_PLAYER_SETPACE：倍速播放状态；

MEDIA_PLAYER_RESUME：恢复播放状态。

C.2.2.4 媒体流类型定义

原型：enum media_track_type {

```
    MEDIA_TRACK_TYPE_UNKNOWN = 0,
    MEDIA_TRACK_TYPE_VIDEO = 1,
    MEDIA_TRACK_TYPE_AUDIO = 2,
    MEDIA_TRACK_TYPE_TIMEDTEXT = 3,
    MEDIA_TRACK_TYPE_SUBTITLE = 4,
};
```

描述：媒体流类型定义。

成员：

MEDIA_TRACK_TYPE_UNKNOWN：未知类型；

MEDIA_TRACK_TYPE_VIDEO：视频类型；

MEDIA_TRACK_TYPE_AUDIO：音频类型；

MEDIA_TRACK_TYPE_TIMEDTEXT：字幕同步类型；

MEDIA_TRACK_TYPE_SUBTITLE：字幕类型。

C.2.3 事件类型

原型：enum media_event_type {

```
    MEDIA_NOP               = 0,
    MEDIA_PREPARED          = 1,
    MEDIA_PLAYBACK_COMPLETE = 2,
    MEDIA_BUFFERING_UPDATE  = 3,
    MEDIA_SEEK_COMPLETE     = 4,
    MEDIA_SET_VIDEO_SIZE    = 5,
    MEDIA_STARTED           = 6,
    MEDIA_PAUSED            = 7,
    MEDIA_STOPPED           = 8,
    MEDIA_SKIPPED           = 9,
    MEDIA_TIMED_TEXT        = 99,
    MEDIA_ERROR             = 100,
    MEDIA_INFO              = 200,
    MEDIA_SUBTITLE_DATA     = 201,
};
```

描述：媒体事件类型定义。

成员：

MEDIA_NOP：不操作事件，用于接口测试；

MEDIA_PREPARED：媒体状态就绪事件；

MEDIA_PLAYBACK_COMPLETE：播放完成状态事件；

MEDIA_BUFFERING_UPDATE：缓冲更新事件；

MEDIA_SEEK_COMPLETE：SEEK完成事件；

MEDIA_SET_VIDEO_SIZE：设置视频大小事件；

MEDIA_STARTED：媒体开始播放事件；

MEDIA_PAUSED：媒体暂停播放事件；

MEDIA_STOPPED：媒体停止播放事件；

MEDIA_SKIPPED：媒体状态跳过事件；

MEDIA_TIMED_TEXT：同步字幕事件；

MEDIA_ERROR：媒体错误事件；

MEDIA_INFO：媒体信息事件；

MEDIA_SUBTITLE_DATA：媒体字幕数据事件。

C.2.4 数据结构

无。

C.2.5 回调函数定义

C.2.5.1 notify回调

原型：void notify(int msg, int ext1, int ext2, const Parcel *obj);

功能：播放器消息通知回调函数。

参数：msg--输入参数，消息类型；
ext1--输入参数，扩展消息类型；
ext2--输入参数，扩展消息类型 2；
obj--输入参数，扩展对象，一般用于传递用户私有数据。
返回：无。

C.2.6 接口定义

C.2.6.1 setDataSource接口

原型：status_t setDataSource(const sp<IMediaHTTPService> &httpService,
const char *url, const KeyedVector<String8, String8> *headers)
功能：以 path 和 Hash Key 对的方式设置数据源
参数：httpService--输出参数，表示使用系统原生 HttpService 获取数据源。该参数在 TVOS 中废弃，后续不建议使用，传入时可以为空；
url--输入参数，数据源 url；
headers--对数据源 uri 的补充说明，唯一限定一个数据源。取值可以为空，为空表示 uri 可唯一确定一个数据源。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.2 setDataSource接口

原型：status_t setDataSource(int fd, int64_t offset, int64_t length);
功能：以文件句柄方式设置数据源。
参数：fd--输入参数，文件句柄；
offset--输入参数，针对文件起始位置偏移量；
length--输入参数，播放长度。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.3 setVideoSurfaceTexture接口

原型：status_t setVideoSurfaceTexture(const sp<IGraphicBufferProducer>& bufferProducer);
功能：设置视频显示画幕。
参数：bufferProducer--输入参数，视频显示画幕。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.4 setListener接口

原型：status_t setListener(const sp<MediaPlayerListener>& listener);
功能：设置媒体播放监听器。
参数：listener--输入参数，播放器监听器。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.5 prepare接口

原型：status_t prepare();
功能：同步的方式让播放器准备，直到播放器准备好才返回，如申请必要资源，缓冲数据等；该接口在 setDataSource 和 setDisplay 之后调用。

参数：无。

返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.6 prepareAsync接口

原型：status_t prepareAsync();

功能：异步的方式让播放器准备，调用后该接口立刻返回，而不需等缓冲数据好；该接口在 setDataSource 和 setDisplay 之后调用。

参数：无。

返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.7 start接口

原型：status_t start();

功能：从 pause 或者 stop 状态启动播放，当之前是 pause 状态时，start 后从之前 pause 位置开始播放，当之前为 stop 时，从数据源的头开始播放。

参数：无。

返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.8 stop接口

原型：status_t stop();

功能：停止播放，在播放结束后调用，或者在 pause 状态时，需要停止播放时调用。

参数：无。

返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.9 pause接口

原型：status_t pause();

功能：暂停播放，暂停播放后可以调用 start 来恢复播放。

参数：无。

返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.10 isPlaying接口

原型：bool isPlaying();

功能：获取播放器状态是否处于播放中。

参数：无。

返回：bool，播放中则返回 true，否则返回 false。

C.2.6.11 getVideoWidth接口

原型：status_t getVideoWidth(int *w);

功能：获取视频播放窗口宽度。

参数：w--int 型指针，表示当前播放媒体中视频窗口的宽度，当没有视频时，值为 0。

返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.12 getVideoHeight接口

原型：status_t getVideoHeight(int *h);
功能：获取视频播放窗口高度。
参数：h--int 型指针，表示当前播放媒体中视频窗口的高度，当没有视频时，值为 0。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.13 setVideoArea接口

原型：status_t setVideoArea (int x, int y, int width, int height);
功能：设置视频播放窗口区域。
参数：x--int 型，窗口左上角 x 坐标；
y--int 型，窗口左上角 y 坐标；
width--int 型，窗口宽度；
height--int 型，窗口高度。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.14 getVideoArea接口

原型：status_t getVideoArea (int *x, int *y, int *width, int *height);
功能：获取视频播放窗口区域。
参数：x--int 型指针，输出参数，输出窗口左上角 x 坐标；
y--int 型指针，输出参数，输出窗口左上角 y 坐标；
width--int 型指针，输出参数，输出窗口宽度；
height--int 型指针，输出参数，输出窗口高度。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.15 seekTo接口

原型：status_t seekTo(int msec);
功能：让播放器在指定的位置播放。
参数：msec--int 类型，表示指定的播放位置。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.16 getCurrentPosition接口

原型：status_t getCurrentPosition(int *msec);
功能：获取当前播放位置。
参数：msec--int 型指针，表示当前的播放位置。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.17 getDuration接口

原型：status_t getDuration(int *msec);
功能：获取播放源的可播放长度，单位 ms。
参数：msec--int 型指针，单位 ms，表示当前数据源可播放长度。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.18 reset接口

原型：status_t reset();
功能：重置播放器，使其处于去初始化状态，后续若仍需要使用播放器，需要先初始化，在设置数据源，然后调用 prepare 让播放器准备就绪。
参数：无。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.19 setPace接口

原型：status_t setPace(int pace);
功能：设置播放器速率，设置后立刻生效，仅用于 VOD 场景。
参数：pace--int 类型，播放倍速。正常播放速率为 1，2 倍快进速率为 2，2 倍速快退速率为-2，以此类推，最大支持到正负 32 倍。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.20 getPace接口

原型：status_t getPace(int * pace);
功能：获取播放器速率。
参数：pace--int 型指针，播放倍速。正常播放速率为 1，2 倍快进速率为 2，2 倍速慢退速率为-2，以此类推，最大支持到正负 32 倍。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.21 setStopMode接口

原型：status_t setStopMode(int mode);
功能：设置媒体播放暂停效果，如静帧、黑屏。
参数：mode--int 型，表示暂停效果，取值如下：
STOP_MODE_BLACK = 0，表示黑屏模式；
STOP_MODE_FREEZE = 1，表示静帧模式。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.22 getStopMode接口

原型：status_t getStopMode(int* mode);
功能：获取媒体播放暂停效果，如静帧、黑屏。
参数：mode--int 指针，表示暂停效果，取值如下：
STOP_MODE_BLACK = 0，表示黑屏模式；
STOP_MODE_FREEZE = 1，表示静帧模式。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.23 setClip接口

原型：status_t setClip(int x, int y, int width, int height);
功能：设置视频源图像的剪切区域。设置后整个视频窗口仅显示剪切区域的源视频，用于实现局部放大缩小等操作，即设置后将指定区域的源视频在整个输出窗口上显示。如果硬件不支持视频剪切，则此方法调用不产生实际效果。
参数：x--int 类型，表示视频剪切区域的横向坐标位置；

y--int 类型，表示视频剪切区域的纵向坐标位置；
width--int 类型，表示剪切区域的宽度；
height--int 类型，表示剪切区域的高度。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.24 getClip接口

原型：status_t getClip(int* x, int* y, int* width, int* height);
功能：获取视频源图像的剪切区域。设置后整个视频窗口仅显示剪切区域的源视频，用于实现局部放大缩小等操作，即设置后将指定区域的源视频在整个输出窗口上显示。如果硬件不支持视频剪切，则此方法调用不产生实际效果。
参数：x--int 型指针，表示视频裁剪区域的横向坐标位置；
y--int 型指针，表示视频裁剪区域的纵向坐标位置；
width--int 型指针，表示裁剪区域的宽度；
height--int 型指针，表示裁剪区域的高度。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.25 getStartTime接口

原型：status_t getStartTime (int64_t *startTime);
功能：获取时移（或者回看）节目的起始时间。
参数：startTime--int64 指针类型，输出参数，输出获取到的起始时间。
返回：status_t，正常则返回 0，startTime 则输出起始时间，否则返回错误码。

C.2.6.26 selectAudioStream接口

原型：status_t selectAudioStream(int audioPid, int audioCodec);
功能：选择当前播放器的音频流，目前仅 DVB/VOD 场景支持。
参数：audioPid--int 类型，媒体流节目 ID；
audioCodec--int 类型，媒体流编码类型。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.27 setLooping接口

原型：status_t setLooping(int loop);
功能：设置播放器是否循环播放，即播放完当前媒体后，重新再播放该媒体。
参数：loop--int 型，0 表示非循环播放，非零表示循环播放。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.28 isLooping接口

原型：bool isLooping();
功能：获取当前状态是否为循环播放，即播放完当前媒体后，重新再播放该媒体；
参数：无。
返回：boolean 类型，ture 表示循环播放，false 表示不循环播放。

C.2.6.29 setDisplayMode接口

原型：status_t setDisplayMode(int eMode);
功能：设置播放器显示模式。
参数：eMode--int 类型，表示显示模式，取值如下：
E_SME_DVBPLAYER_DIS_MODE_FULL = 0，表示全屏；
E_SME_DVBPLAYER_DIS_MODE_FITIN = 1，表示原始比例全屏显示，左右/上下留黑边；
E_SME_DVBPLAYER_DIS_MODE_FITOUT = 2，表示原始比例全屏显示，左右/上下输出到屏幕外；
E_SME_DVBPLAYER_DIS_MODE_FITCENTER = 3，表示原始比例显示在屏幕中间，左右/上下黑边。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.30 setDisplayRatio接口

原型：status_t setDisplayRatio(int eRatio);
功能：设置播放时视频的宽高比率。
参数：eRatio--int 类型，表示显示窗口宽高比，取值如下：
DISPLAY_RATIO_4TO3 = 0，表示宽高为 4 比 3；
DISPLAY_RATIO_16TO9 = 1，表示宽高为 16 比 9；
DISPLAY_RATIO_AUTO = 6，表示宽高为自动选择。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.31 setMetadataFilter接口

原型：status_t setMetadataFilter(const Parcel& filter);
功能：设置媒体元数据过滤器。
参数：filter--Parcel 容器，表示过滤器对象。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.32 getMetadata接口

原型：status_t getMetadata(bool update_only, bool apply_filter, Parcel *metadata);
功能：获取媒体元数据。
参数：update_only--boolean 类型，表示是否返回全部的 MetaData 对象，ture 表示全部，false 表示只返回最后一次更新的 MetaData 数据；
apply--boolean 类型，表示是否使用过滤器，false 表示不使用，true 表示使用，此时会根据之前调用 setMetadataFilter 接口所使用的过滤条件过滤；
metadata--Parcel 对象，表示获取到的媒体元数据，当出错时返回 NULL；当未获取到有效 MetaData 时，返回空的 MetaData 对象。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.33 setVideoDisplay接口

原型：status_t setVideoDisplay(int flag);
功能：设置视频输出。
参数：flag--int 类型，表示视频显示是关闭、还是打开，取值如下：
TVOS_VIDEO_DISPLAY_CLOSE = 0，表示关闭；
TVOS_VIDEO_DISPLAY_OPEN = 1，表示打开。
返回：status_t，正常则返回 0，否则返回错误码。

C.2.6.34 getVideoDisplay接口

原型：status_t getVideoDisplay(int *flag);

功能：获取视频输出状态。

参数：flag--int 指针类型，输出参数，表示视频显示是关闭、还是打开，取值如下：

TVOS_VIDEO_DISPLAY_CLOSE = 0，表示关闭；

TVOS_VIDEO_DISPLAY_OPEN = 1，表示打开。

返回：status_t，获取成功返回 0，否则返回错误码。

附 录 D
（规范性附录）
HTML5 引擎组件

D.1 概述

本附录定义了 HTML5 引擎组件对外提供的接口，包括 HTML5 功能模块对外接口，见表 D.1。HTML5 页面解析及显示应符合 W3C HTML 5.2、W3C CSS 2.1 及 W3C DOM 2.1。

表 D.1 HTML5 引擎组件功能模块

模块	说明
HTML5 功能模块	定义了 HTML5 页面控制相关功能接口。

D.2 HTML5 功能模块

D.2.1 概述

本模块定义了HTML5页面控制相关功能接口，见表D.2。

表 D.2 HTML5 功能模块接口

接口	说明
browser_main	启动浏览器服务。
page_open	打开一个新的页面。
page_load	加载网址。
page_reload	刷新当前页面。
page_stop	停止加载当前页面。
page_backForward	对当前页面进行前进后退操作。
page_close	关闭当前页面。
page_setPageCreateCallback	设置页面创建完成回调函数。
page_setStartLoadCallback	设置页面开始加载回调函数。
page_setFinishLoadCallback	设置页面结束加载回调函数。
page_setFailLoadCallback	设置页面加载失败回调函数。

D.2.2 常量定义

D.2.2.1 打开页面返回错误码

原型：#define HWB_OK 0
描述：打开页面正常。
原型：#define HWB_ERROR_PAGE_EXIST 1
描述：打开页面错误，页面已存在。
原型：#define HWB_ERROR_PAGE_NOT_EXIST 2
描述：打开页面错误，页面不存在。

原型：#define HWB_ERROR_PAGE_NAME_INVALIDATE 3
描述：打开页面错误，页面名称无效。
原型：#define HWB_ERROR_PAGENUM_EXCEED_MAX_LIMIT 4
描述：打开页面错误，页面个数超过最大个数。
原型：#define HWB_ERROR_EXCEED_RECT 5
描述：打开页面错误，页面区域无效。

D.2.2.2 打开页面其它常量

原型：#define HWB_PAGE_NAME_SIZE 100
描述：页面名称最大字符串长度。
原型：#define HWB_PAGE_MAX_NUM 8
描述：容许打开页面最大数。
原型：#define HWB_DEFAULT_PAGE_NAME "DefaultPage"
描述：默认页面名称。
原型：#define HWB_DEFAULT_URL "about:blank"
描述：默认页面Url。

D.2.3 事件类型

无。

D.2.4 数据结构

无。

D.2.5 回调函数定义

D.2.5.1 Notify_pageCreate回调

原型：typedef int(*Notify_pageCreate)(const char* pageName, int pageStatus);
功能：页面创建完成的回调函数。
参数：pageName—新创建页面的页面名称；
pageStatus--新创建页面的页面状态。
返回：int，-1 表示失败，其他表示成功。

D.2.5.2 Notify_startLoad回调

原型：typedef int(*Notify_startLoad)(const char* pageName, const char* url);
功能：页面开始加载的回调函数。
参数：pageName—对应页面的页面名称；
url—对应页面开始加载的网址。
返回：int，-1 表示失败，其他表示成功。

D.2.5.3 Notify_finishLoad回调

原型：typedef int(*Notify_finishLoad)(const char* pageName, const char* url);
功能：页面完成加载的回调函数。
参数：pageName—对应页面的页面名称；

url—对应页面完成加载的网址。

返回：int，-1 表示失败，其他表示成功。

D.2.5.4 Notify_failLoad回调

原型：typedef int(*Notify_failLoad)(const char* pageName, const char* url, int errorCode, const char* description);

功能：页面加载失败的回调函数。

参数：pageName—对应页面的页面名称；

url—对应页面加载失败的网址。

errorCode—对应页面加载失败的错误码。

description—对应页面加载失败的简易描述。

返回：int，-1 表示失败，其他表示成功。

D.2.6 接口定义

D.2.6.1 browser_main接口

原型：int browser_main(int argc, const char** argv);

功能：启动浏览器服务。

参数：argc—参数个数；

argv—启动参数字符串数组首地址，启动参数取值见表 D.3。

表 D.3 Browser_main 启动参数

参数名	说明
single-process	单进程模式启动
disable-gpu	禁止使用 GPU
enable-spatial-navigation	允许使用平面焦点导航，用方向键可以切换 a 标签的焦点位置
allow-file-access-from-files	允许 file://协议方式访问页面
ui-disable-threaded-compositing	禁止用户交互线程合成
no-zygote	不需要孵化进程
no-sandbox	停用沙箱
disable-web-security	不遵守同源策略
disable-accelerated-video	停用 GPU 加速视频
disable-webaudio	禁止使用 Web 音频 API
disable-plugins	禁用插件
ozone-haisi-init-param	Ozone 平台启动时设置的参数
disable-setuid-sandbox	禁止使用 setuid 沙箱
ozone-platform	配置 Ozone 平台的名字，如：directfb
enable-logging	使用日志
log-level	设置日志打印级别，INFO = 0, WARNING = 1, LOG_ERROR = 2, LOG_FATAL = 3
blink-platform-log-channels	设置平台日志通道，例如: Media

表 D.3（续）

参数名	说明
allow-sandbox-debugging	允许调试沙箱
crash-test	崩溃测试
debug-plugin-loading	转储额外的日志记录插件加载到日志文件
disable-accelerated-2d-canvas	禁用 GPU 加速的 2D 画布
disable-application-cache	禁用应用程序缓存
disable-databases	禁用 HTML5 数据库
disable-webgl	禁用 WebGL
disable-file-system	禁用文件系统 API
disable-local-storage	禁用本地存储
disable-logging	禁用日志
disable-media-source	禁用媒体源 API （例如 MediaSource 对象）
disable-renderer-accessibility	关闭渲染访问
disable-session-storage	禁用会话存储
disable-shared-workers	禁用共享工具
disable-svg1dom	禁用 SVG 1.1 DOM
disable-xslt	禁用 XSLT
enable-encrypted-media	使用加密媒体扩展
v8-cache-options	设置 v8 缓存选项，如 off, preparse data,或者 code
enable-tracing	在浏览器测试的执行过程中启用跟踪
enable-tracing-output	写入测试跟踪的输出文件的文件名
enable-v8-script-streaming	使流脚本 V8 同时加载
extra-plugin-dir	从指定的目录加载 NPAPI 插件

返回：int，0 表示成功，其他表示失败。

D.2.6.2 page_open接口

原型：int page_open(const char* pageName, const char* url, int x, int y, int width, int height);

功能：打开一个新的页面。

参数：pageName—新页面的名称；

url—新页面的地址。

x—新页面的横坐标，以像素为单位。

y—新页面的纵坐标，以像素为单位。

width—新页面的宽度，以像素为单位。

height—新页面的高度，以像素为单位。

返回：int，0表示成功，其他表示失败。

D.2.6.3 page_load接口

原型：int page_load(const char* pageName, const char* url);

功能：加载网址。
参数：pageName—页面的名称；
　　url—要加载的地址。
返回：int，0表示成功，其他表示失败。

D.2.6.4 page_reload接口

原型：int page_reload(const char* pageName);
功能：刷新当前页面。
参数：pageName—页面的名称；
返回：int，0表示成功，其他表示失败。

D.2.6.5 page_stop接口

原型：int page_stop(const char* pageName);
功能：停止加载当前页面。
参数：pageName—页面的名称；
返回：int，0表示成功，其他表示失败。

D.2.6.6 page_backForward接口

原型：int page_backForward(const char* pageName, int offset);
功能：对当前页面进行前进后退操作。
参数：pageName—页面的名称；
　　offset—负数表示后退，正数表示前进
返回：int，0表示成功，其他表示失败。

D.2.6.7 page_close接口

原型：int page_close(const char* pageName);
功能：关闭当前页面。
参数：pageName—页面的名称；
返回：int，0表示成功，其他表示失败。

D.2.6.8 page_setPageCreateCallback接口

原型：int page_setPageCreateCallback(Notify_PageCreate func);
功能：设置页面创建完成回调函数。
参数：func—对应的回调函数；
返回：int，0表示成功，其他表示失败。

D.2.6.9 page_setStartLoadCallback接口

原型：int page_setStartLoadCallback(Notify_startLoad func);
功能：设置页面开始加载回调函数。
参数：func—对应的回调函数；
返回：int，0表示成功，其他表示失败。

D.2.6.10 page_setFinishLoadCallback接口

原型：int page_setFinishLoadCallback(Notify_finishLoad func);
功能：设置页面结束加载回调函数。
参数：func—对应的回调函数；
返回：int，0表示成功，其他表示失败。

D.2.6.11 page_setFailLoadCallback接口

原型：int page_setFailLoadCallback(Notify_failLoad func);
功能：设置页面加载失败回调函数。
参数：func—对应的回调函数；
返回：int，0 表示成功，其他表示失败。

附 录 E
（规范性附录）
DRM 组件

E.1 概述

本附录定义了 DRM 组件对外提供的接口，包括 DRM 功能模块对外接口,见表 E.1。

表 E.1 DRM 组件功能模块

模块	说明
DRM 功能模块	定义了 DRM 相关功能接口。

E.2 DRM功能模块

E.2.1 概述

本模块定义了DRM相关功能接口，见表E.2。

表 E.2 DRM 功能模块接口

接口	说明
CHDRMAPI_RegisterApp	注册。
CHDRMAPI_RegisterApp	注册拓展，可以自定义私有的与 TApp 通信的查询许可证和解密命令 ID。
CHDRMAPI_UnRegisterApp	注销。
CHDRMAPI_SendCommandToTEE	发送命令到 TEE。
CHDRMAPI_SendMessageToPlayer	通知播放器消息。
CHDRMAL_CreateCryptoSession	创建解密会话。
CHDRMAL_DestroyCryptoSession	销毁解密会话。
CHDRMAL_UpdateCryptoSession	更新 DRM 信息。
CHDRMAL_CheckRights	查询许可证信息。
CHDRMAL_Decrypt	内容解密。
CHDRMAPI_SetLicenseReq_Callback	注册许可证监听函数。
CHDRMAPI_SetDecryptReq_Callback	注册解密监听函数。
CHDRMAPI_SetMessage_Callback	注册消息监听函数。
CHDRMAL_SetPlayerEventListenser	注册播放器时间监听函数。

E.2.2 常量定义

E.2.2.1 处理成果

原型：#define CHDRM_NO_ERROR 0
描述：处理成功。

E.2.3 事件类型

无。

E.2.4 数据结构

E.2.4.1 DRM解密类型

原型：typedef enum {
    DECRYPT_LICENSEREQ_APP_INVOKE = 0x00,
    DECRYPT_LICENSEREQ_MANAGER_INVOKE = 0x01,
    DECRYPT_APP_LICENSEREQ_MANAGER_INVOKE = 0x02,
    DECRYPT_MANAGER_LICENSEREQ_APP_INVOKE = 0x03,
} DRM_DECRYPT_FLAG_E;
描述：解密类型定义。
成员：
DECRYPT_LICENSEREQ_APP_INVOKE：解密数据且查询许可证使用回调函数通知DRM APP调用；
DECRYPT_LICENSEREQ_MANAGER_INVOKE：解密数据且查询许可证DRM Manager直接sendDatatoTee；
DECRYPT_APP_LICENSEREQ_MANAGER_INVOKE：解密数据使用回调函数通知DRM APP调用，查询许可证DRM Manager直接sendDatatoTee；
DECRYPT_MANAGER_LICENSEREQ_APP_INVOKE：解密数据DRM Manager直接sendDatatoTee，查询许可证使用回调函数通知DRM APP调用。

E.2.4.2 TEE的返回数据结构

原型：typedef struct ChinaDrm_TeeRetVal_t {
    unsigned char *data;
    unsigned long dataLen;
    unsigned int originCode;
    unsigned int returnCode;
} ChinaDrm_TeeRetVal;
描述：TEE的返回参数定义。
成员：
data：返回数据；
dataLen：返回数据长度；
originCode：原始码；
returnCode：返回码。

E.2.4.3 DRM节目描述数据结构

原型：typedef struct ChinaDrm_Info_t {
    unsigned char* drm_id;
    unsigned char *keydata;
    unsigned int keydata_len;
} ChinaDrm_Info;
描述：ChinaDRM信息结构定义。
成员：
drm_id：DRM APP唯一标识；

keydata：m3u8数据；
keydata_len：m3u8数据长度。

E.2.4.4 DRM节目解密数据结构

原型：typedef struct Crypto_Info_t {
    unsigned char *segmentkeydata;
    unsigned int Segmentkeydata_len;
} Crypto_Info;
描述：解密信息结构定义。
成员：
segmentkeydata：Segment数据；
Segmentkeydata_len：Segment数据长度。

E.2.5 回调函数定义

E.2.5.1 DRM_LicenseReq_Callback回调

原型：typedef int (*DRM_LicenseReq_Callback)(unsigned char* data, unsigned int data_len);
功能：设置获取许可证的回调函数。
参数：data--许可证内容数据指针；
      data_len--许可证内容数据长度。
返回：int，0表示成功，其他表示失败。

E.2.5.2 DRM_DecryptReq_Callback回调

原型：typedef int (*DRM_DecryptReq_Callback)(long int enc_phyaddr,
      unsigned int enc_data_len, long int dec_phyaddr,
      unsigned char* extension, unsigned int extension_len);
功能：设置数据解密的回调函数。
参数：enc_phyaddr—加密数据物理地址；
      enc_data_len--加密数据长度；
      dec_phyaddr—解密数据物理地址；
      extension—扩展内容指针；
      extension_len—扩展内容长度。
返回：int，0表示成功，其他表示失败。

E.2.5.3 DRM_Message_Callback回调

原型：typedef int (*DRM_Message_Callback)(int type, void* message, int len);
功能：设置消息通知的回调函数。
参数：type—消息类型；
      message--消息内容指针；
      len—消息内容长度。
返回：int，0表示成功，其他表示失败。

E.2.5.4 ChinaDrmMessageEvent回调

原型：typedef int (*ChinaDrmMessageEvent)(int type, void* message, int len);
功能：设置播放器处理 DRM 事件的事件监听器。
参数：type—消息类型；
message--消息内容指针；
len—消息内容长度。
返回：int，0 表示成功，其他表示失败。

E.2.6 接口定义

E.2.6.1 CHDRMAPI_RegisterApp接口

原型：int CHDRMAPI_RegisterApp(unsigned char* drm_id, unsigned char* uuid,
unsigned int uuidLen, int register_commandid,
const unsigned char* register_pridata,DRM_DECRYPT_FLAG_E enflag);
功能：DRM APP 注册。
参数：drm_id--DRM APP 唯一标识；
uuid--DRM APP 对应 TApp 的唯一标识；
uuidLen--uuid 的长度；
register_commandid--与 TApp 通信的注册 commandid；
register_pridata--与 TApp 通信的注册携带的私有注册数据,用于 TApp 验证 DRM APP 合法性；
enflag--解密调用方式。
返回：int，0表示成功，其他表示失败。

E.2.6.2 CHDRMAPI_RegisterApp接口

原型：int CHDRMAPI_RegisterApp(unsigned char* drm_id, unsigned char* uuid,
unsigned int uuidLen, int register_commandid,
const unsigned char* register_pridata,int enflag,
int licensereq_commandid, int decrypt_commandid);
功能：DRM APP 注册扩展，可以自己私有定义 licensereq、decrypt 设置 TApp 的命令 ID。
参数：drm_id--DRM APP 唯一标识；
uuid--DRM APP 对应 TApp 的唯一标识；
uuidLen--uuid 的长度；
register_commandid--与 TApp 通信的注册 commandid；
register_pridata--与 TApp 通信的注册携带的私有注册数据,用于 TApp 验证 DRM APP 合法性；
enflag--解密调用方式；
licensereq_commandid--查询许可证对应的 commandid；
decrypt_commandid--解密数据对应的 commandid。
返回：int，0表示成功，其他表示失败。

E.2.6.3 CHDRMAPI_UnRegisterApp接口

原型：int CHDRMAPI_UnRegisterApp(void);
功能：DRMAPP 注销。
参数：无。
返回：int，0表示成功，其他表示失败。

E.2.6.4 CHDRMAPI_SendCommandToTEE接口

原型：int CHDRMAPI_SendCommandToTEE(unsigned int commandId,
unsigned char* inputData, unsigned int inputDataLen,
ChinaDrm_TeeRetVal* outputData);
功能：发送命令到TEE。
参数：commandId--命令ID;
inputData--发送的数据;
inputDataLen--发送数据的长度;
outputData--返回的数据信息。
返回：int，0表示成功，其他表示失败。

E.2.6.5 CHDRMAPI_SendMessageToPlayer接口

原型：int CHDRMAPI_SendMessageToPlayer(int type, const char* message, int message_len);
功能：发送消息到播放器。
参数：type--消息类型;
message--消息;
message_len--消息长度。
返回：int，0表示成功，其他表示失败。

E.2.6.6 CHDRMAL_CreateCryptoSession接口

原型：int CHDRMAL_CreateCryptoSession(ChinaDrmCrypto_Handle* crypto_handle,
ChinaDrm_Info* drminfo);
功能：创建解密句柄。
参数：crypto_handle--解密句柄;
drminfo--drmInfo信息。
返回：int，0表示成功，其他表示失败。

E.2.6.7 CHDRMAL_DestroyCryptoSession接口

原型：int CHDRMAL_DestroyCryptoSession(ChinaDrmCrypto_Handle crypto_handle);
功能：销毁解密句柄。
参数：crypto_handle--解密句柄。
返回：int，0表示成功，其他表示失败。

E.2.6.8 CHDRMAL_UpdateCryptoSession接口

原型：int CHDRMAL_UpdateCryptoSession(ChinaDrmCrypto_Handle crypto_handle,
ChinaDrm_Info* drminfo);
功能：更新DRM信息。
参数：crypto_handle--解密句柄;
drminfo--drmInfo信息。
返回：int，0表示成功，其他表示失败。

E.2.6.9 CHDRMAL_CheckRights接口

原型：int CHDRMAL_CheckRights(ChinaDrmCrypto_Handle crypto_handle,
Crypto_Info* cryptoinfo);
功能：查询内容是否有授权。
参数：crypto_handle--解密句柄；
cryptoinfo --cryptoInfo 信息。
返回：int，0表示成功，其他表示失败。

E.2.6.10 CHDRMAL_Decrypt接口

原型：int CHDRMAL_Decrypt(ChinaDrmCrypto_Handle crypto_handle,
Crypto_Info* cryptoinfo, unsigned char* encbuf, unsigned int encbuf_len,
unsigned char* decbuf, unsigned int decbuf_len, unsigned char* iv,
unsigned int iv_len, Mode mode,Const SubSample* subSamples,
int numSubSamples);
功能：解密数据。
参数：crypto_handle--解密句柄；
cryptoinfo--解密信息；
encbuf--加密的数据；
encbuf_len--加密的数据长度；
decbuf--解密后的数据存放地址；
ecbuf_len--解密后的数据长度；
iv--解密 IV；
iv_len--解密 IV 长度；
mode--解密算法标示；
subSamples--加密信息描述；
numSubSamples 加密单元数量。
返回：int，0 表示成功，其他表示失败。

E.2.6.11 CHDRMAPI_SetLicenseReq_Callback接口

原型：int CHDRMAPI_SetLicenseReq_Callback(DRM_LicenseReq_Callback callback);
功能：设置获取许可证的回调函数。
参数：callback--获取许可证的回调函数。
返回：int，0 表示成功，其他表示失败。

E.2.6.12 CHDRMAPI_SetDecryptReq_Callback接口

原型：int CHDRMAPI_SetDecryptReq_Callback(DRM_DecryptReq_Callback callback);
功能：设置数据解密的回调函数。
参数：callback--数据解密的回调函数。
返回：int，0表示成功，其他表示失败。

E.2.6.13 CHDRMAPI_SetMessage_Callback接口

原型：int CHDRMAPI_SetMessage_Callback(DRM_Message_Callback callback)
功能：设置消息通知的回调函数。

参数：callback--消息通知的回调函数。

返回：int，0表示成功，其他表示失败。

E.2.6.14 CHDRMAL_SetPlayerEventListenser接口

原型：int CHDRMAL_SetPlayerEventListenser(ChinaDrmMessageEvent callback);

功能：设置播放器处理DRM事件的事件监听器。

参数：callback--事件监听回调函数。

返回：int，0 表示成功，其他表示失败。

附 录 F
（规范性附录）
DCAS 组件

F.1 概述

本附录定义了 DCAS 组件对外提供的接口，包括 CA 应用功能模块和解扰操作功能模块对外接口，见表 F.1。

表 F.1 DCAS 组件功能模块

模块	说明
CA 应用功能模块	定义了支撑 DCAS 应用的功能接口。
解扰操作功能模块	定义了解扰操作相关功能接口。

F.2 CA应用功能模块

F.2.1 概述

本模块定义了支撑DCAS应用的功能接口,见表F.2。

表 F.2 CA 应用功能模块接口

接口	说明
DCASAL_registerCASModule	注册 DCAS 回调函数。
DCASAL_removeCASModule	删除 DCAS 模块的注册。
DCASAL_enableDescramblingRequests	启动接收解扰请求。
DCASAL_disableDescramblingRequests	停止接收解扰请求。
DCASAL_startEcmLoading	启动接收 ECM。
DCASAL_stopEcmLoading	停止接收 ECM。
DCASAL_startInbandEmmLoading	启动接收带内 EMM。
DCASAL_stopInbandEmmLoading	停止接收带内 EMM。
DCASAL_sendCommandToTEE	发送命令到 TEE 中。
DCASAL_getChipPublicId	获取 ChipId。
DCASAL_sendDescramblingEvent	发送解扰状态。
DCASAL_setData	存储数据接口。
DCASAL_getDSD	获得 DSD 句柄。
DCASAL_registerDSDListener	注册 DSDListener。
DCASAL_removeDSDListener	删除 DSDListener。
DCASAL_getData	读取数据接口。
DCASAL_setCAInfo	设置接口。
DCASAL_getCAInfo	查询接口。

F.2.2 常量定义

F.2.2.1 会话参数

原型：#define DCAS_CASDESCRIPTOR_PRIVATEDATA_MAX_LEN 32
描述：相关私有描述符最大的长度。
原型：#define DCAS_CASSESSION_STREAM_MAX_LEN 8
描述：会话的音视频节目最大个数。
原型：#define DCAS_CASSESSION_STREAM_PATH_MAX_LEN 8
描述：会话的安全视频路径最大个数。

F.2.2.2 触发ECM事件的数据长度

原型：#define DCAS_CASEMMEVENT_DATA_MAX_LEN 204
描述：ECM 数据最大长度。

F.2.2.3 触发EMM事件的数据长度

原型：#define DCAS_CASECMEVENT_DATA_MAX_LEN 204
描述：ECM 数据最大长度。

F.2.2.4 过滤器参数的数据长度

原型：#define DCAS_FILTERMASK_VALUE_MAX_LEN 32
描述：过滤器参数的最大长度。

F.2.2.5 返回芯片ID参数的数据长度

原型：#define DCAS_CHIPID_VALUE_MAX_LEN 32
描述：芯片 ID 参数的最大长度。

F.2.3 事件类型

无。

F.2.4 数据结构

F.2.4.1 会话数据结构

原型：typedef struct DCAS_CASSession {

```
    DCAS_CAData_t casDescriptor;
    unsigned int channelNumber;
    unsigned int networkID;
    unsigned int oprationType;
    unsigned int programNumber;
    unsigned int serviceIdentifier;
    unsigned int sessionID;
    unsigned int streamPath[DCAS_CASSESSION_STREAM_PATH_MAX_LEN];
    unsigned short streamPIDs[DCAS_CASSESSION_STREAM_MAX_LEN];
    unsigned short streamTypes[DCAS_CASSESSION_STREAM_MAX_LEN];
    unsigned int transmitterScramblingMode;
```

```
    unsigned int transportStreamID;
    unsigned int tunerId;
} DCAS_CASSession_t;
```

描述：CAS的Session参数定义。

成员：

casDescriptor：解复用器ID；

channelNumber：频道号；

networkID：网路ID；

oprationType：运营商类型；

programNumber：节目号；

serviceIdentifier：服务ID；

sessionID：会话ID；

streamPath：定长3个int类型的安全视频路径参数；

streamPIDs：最大支持8个PID；

streamTypes：最大支持8个音视频类型和上面一一对应；

transmitter_scramblingmode：码流CW加扰模式；

transportStreamID：传输流ID；

tuner_id：Tuner的ID。

F.2.4.2 返回数据结构

```
原型：typedef struct DCAS_TeeRetVal {
    unsigned char *data;
    unsigned long dataLen;
    unsigned int originCode;
    unsigned int returnCode;
} DCAS_TeeRetVal_t;
```

描述：TEE的返回参数定义。

成员：

data：TEE返回的数据缓冲区指针；

dataLen：TEE返回数据长度；

originCode：TEE返回的原始码；

returnCode：TEE返回的错误码。

F.2.4.3 ECM事件数据结构

```
原型：typedef struct DCAS_CASEcmEvent {
    unsigned char ecmdata[DCAS_CASECMEVENT_DATA_MAX_LEN];
    unsigned int error;
    unsigned char tableID;
    unsigned char isTimeout;
} DCAS_CASEcmEvent_t;
```

描述：CAS的ECM事件的参数定义。

成员：

ecmdata：ECM数据；
error：错误码；
tableID：表ID；
isTimeout：ECM包是否超时。

F.2.4.4 EMM事件数据结构

```
原型：typedef struct DCAS_CASEmmEvent {
    unsigned char emmdata[DCAS_CASEMMEVENT_DATA_MAX_LEN];
    unsigned int error;
    unsigned char tableID;
    unsigned char isCatUpdateNotification;
} DCAS_CASEmmEvent_t;
```

描述：CAS的EMM事件的参数定义。
成员：
emmdata：EMM数据；
error：错误码；
tableID：表ID；
isTimeout：ECM包是否超时。

F.2.4.5 DTV过滤器数据结构

```
原型：typedef struct DCAS_FilterParam {
    unsigned short tableId;
    unsigned char offset;
    unsigned char filter[DCAS_FILTERMASK_VALUE_MAX_LEN];
    unsigned char filterLen;
    unsigned char mask[DCAS_FILTERMASK_VALUE_MAX_LEN];
    unsigned char maskLen;
} DCAS_FilterParam_t;
```

描述：TS过滤器的参数定义。
成员：
tableId：表ID；
offset：过滤器偏移；
filter：过滤器数据；
filterLen：过滤器数据长度；
mask：过滤器掩码；
maskLen：过滤器掩码长度。

F.2.4.6 CAS状态数据结构

```
原型：typedef struct DCAS_CASStatus {
    unsigned int casToken;
    unsigned int statusData[DCAS_CASSTATUS_STATUSDATA_MAX_LEN];
    unsigned char isSuccess;
```

```
    unsigned int majorContentProblem;
} DCAS_CASStatus_t;
```

描述：CAS状态信息的参数定义。

成员：

casToken：状态的token标记；

statusData：状态数据；

isSuccess：是否成功；

majorContent：主要内容。

F.2.4.7 芯片ID数据结构

```
原型：typedef struct DCAS_ChipId {
    unsigned char value[DCAS_CHIPID_VALUE_MAX_LEN];
    unsigned char len;
} DCAS_ChipId_t;
```

描述：CAS的芯片ID的参数定义。

成员：

value：长度见F.2.2.5 DCAS_CHIPID_VALUE_MAX_LEN；

len：chipid的长度。

F.2.5 回调函数定义

F.2.5.1 DCAS_StartDescrambling回调

```
原型：typedef int (*DCAS_StartDescrambling)(unsigned int casId,
      DCAS_CASSession_t* casSession, DCAS_CASEcmEvent_t* firstEcmData);
```

功能：开始解扰的回调函数。

参数：casId--CA的SystemID；

casSession--解扰的session参数指针；

firstEcmData--解扰的ECM事件函数指针。

返回：int，0表示成功，其他表示错误。

F.2.5.2 DCAS_UpdateDescrambling回调

```
原型：typedef int (*DCAS_UpdateDescrambling)(unsigned int casId,
      DCAS_CASSession_t* casSession);
```

功能：更新解扰的回调函数。

参数：casId--CA的SystemID；

casSession--解扰的session参数指针。

返回：int，0表示成功，其他表示错误。

F.2.5.3 DCAS_StopDescrambling回调

```
原型：typedef int (*DCAS_StopDescrambling)(unsigned int casId,
      DCAS_CASSession_t* casSession);
```

功能：停止解扰的回调函数。

参数：casId--CA的SystemID；

casSession--解扰的session参数指针。

返回：int，0表示成功，其他表示错误。

F.2.5.4 DCAS_FireEcmEvent回调

原型：typedef int (*DCAS_FireEcmEvent)(unsigned int casId,
DCAS_CASSession_t* casSession, DCAS_CASEcmEvent_t* ecmEvent);

功能：触发ECM的回调函数。

参数：casId--CA的SystemID；
casSession--解扰的session参数指针；
ecmEvent--解扰的ECM事件函数指针。

返回：int，0表示成功，其他表示错误。

F.2.5.5 DCAS_FireInbandEmmEvent回调

原型：typedef int (*DCAS_FireInbandEmmEvent)(unsigned int casId,
DCAS_CASSession_t* casSession, DCAS_CASEmmEvent_t* emmEvent);

功能：触发EMM的回调函数。

参数：casId--CA的SystemID；
casSession--解扰的session参数指针；
emmEvent--解扰的EMM事件函数指针。

返回：int，0表示成功，其他表示错误。

F.2.5.6 DSDStatus回调

原型：typedef void (*DSDStatus)(int status);

功能：DSD状态变化回调函数。

参数：status—状态。

返回：无。

F.2.6 接口定义

F.2.6.1 DCASAL_registerCASModule接口

原型：int DCASAL_registerCASModule(long vendorId,unsigned int casId,int networkPriority,
DCAS_StartDescrambling onStartDescrambling,
DCAS_UpdateDescrambling onUpdateDescrambling,
DCAS_StopDescrambling onStopDescrambling);

功能：本方法是DCAS框架层调用DCAS组件的接口，注册DCAS回调函数。

参数：vendorId—分配给DCAS厂商的ID；
casId—CASModule管理的caSystemId；
networkPriority—用于在超过一个CASModule在CASModuleManager中注册时使用，运营商可根据每个CASModule决定是否该参数可选，当优先级策略启用时，为运营商需要为每个CASModule指定优先级。终端软件平台应选择已注册，具有最高优先级，并且所管理的caSystemId在PMT中有相应CA描述符的CASModule发出解

扰请求。当优先级策略禁用时，DCAS应用应给该参数置零，CASModule的决定方法由终端软件平台自行实现；

onStartDescrambling—开始解扰回调函数；

onUpdateDescrambling—更新解扰回调函数；

onStopDescrambling—停止解扰回调函数。

返回：int，0表示成功，其他表示错误。

F.2.6.2 DCASAL_removeCASModule接口

原型：int DCASAL_removeCASModule(long vendorId,unsigned int casId);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，删除 DCAS 模块的注册。

参数：vendorId--分配给 DCAS 厂商的 ID；

casId--CASModule 管理的 caSystemId。

返回：int，0 表示成功，其他表示错误。

F.2.6.3 DCASAL_enableDescramblingRequests接口

原型：int DCASAL_enableDescramblingRequests(unsigned int casId,
unsigned int firstEcmTimeout, unsigned char autoLoadFirstEcm,
unsigned char isFastMode, unsigned int* ecmTableIds,
unsigned int ecmTableIdsNum, DCAS_FireEcmEvent onEcmEvent);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法启动接收解扰请求。

参数：casId--CASModule 管理的 caSystemId；

firstEcmTimeout--单位毫秒，平台等待第一个 ECM 的最长时间，如果超时，CAS 模块会通过 onEcmEvent 调用或者 onSTartDescrambling 调用收到 CASEcmEvent；

autoLoadFirstEcm--指定是否 auto-load 模式，auto-load 模式是指平台在 JS DCAS 应用调用这个方法后，自动开始过滤第一个 ECM，而不必等待 JS DCAS 应用调用 startEcmLoading；

isFastMode--快速模式（暂时仅是占位，并没有实际意义）；

ecmTableIds--如果 JS DCAS 应用希望指定多个 ECM 的 tableId；

ecmTableIdsNum--ecmTableIds 的数量；

onEcmEvent--回调函数，接收 ECM 数据。

返回：int，0 表示成功，其他表示错误。

F.2.6.4 DCASAL_disableDescramblingRequests接口

原型：int DCASAL_disableDescramblingRequests(unsigned int casId);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法停止接收解扰请求。

参数：casId--CASModule 管理的 caSystemId。

返回：int，0 表示成功，其他表示错误。

F.2.6.5 DCASAL_startEcmLoading接口

原型：int DCASAL_startEcmLoading(unsigned int casId, DCAS_CASSession_t* casSession,
DCAS_FireEcmEvent onEcmEvent);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法启动接收特定加扰节目的 ECM，

此方法在收到解扰请求后调用，在 auto-load 模式中，不需要调用此方法。当有 ECM 数据时，通过 onEcmEvent 来接收数据。

参数：casId--CASModule 管理的 caSystemId；
casSession--从 CASModule.onStartDescrambling 获得的 casSession；
onEcmEvent--回调函数，接收 ECM 数据。

返回：int，0 表示成功，其他表示错误。

F.2.6.6 DCASAL_stopEcmLoading接口

原型：int DCASAL_stopEcmLoading(unsigned int casId, DCAS_CASSession_t* casSession);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法停止接收 ECM。

参数：casId--CASModule 管理的 caSystemId；
casSession--从 onStartDescrambling 回调函数获得的 casSession。

返回：int，0 表示成功，其他表示错误。

F.2.6.7 DCASAL_startInbandEmmLoading接口

原型：int DCASAL_startInbandEmmLoading(unsigned int casId, DCAS_FilterParam_t* filterParm,
int filterParmNum, unsigned char includeCatNotifications,
DCAS_FireInbandEmmEvent onInbandEmmEvent);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法启动接收带内 EMM，当有 EMM 时，通过 onInbandEmmEvent 来接收数据。

参数：casId--CASModule 管理的 caSystemId；
filterParm--过滤器参数，见结构体；
filterParmNum--过滤器个数；
includeCatNotifications--指定是否接收 CAT 信息；
onInbandEmmEvent--回调函数，接收 EMM 数据。

返回：int，0 表示成功，其他表示错误。

F.2.6.8 DCASAL_stopInbandEmmLoading接口

原型：int DCASAL_stopInbandEmmLoading(unsigned int casId);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法停止接收带内 EMM。

参数：casId--CASModule 管理的 caSystemId。

返回：int，0 表示成功，其他表示错误。

F.2.6.9 DCASAL_sendCommandToTEE接口

原型：int DCASAL_sendCommandToTEE(unsigned char* uuid,unsigned int uuidLen,
unsigned char commandId, unsigned char* inputData, unsigned int inputDataLen,
unsigned int applicationContext, DCAS_TeeRetVal_t* outputData);

功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法把命令发送给 TEE 中。

参数：uuid--TA 的 UUID，每个 DCAS 厂家都有不同的 ID；
uuidLen--uuid 的长度；
commandId--TEE 通信中的 ID，ID 值代表的含义由各 DCAS 厂家自己定义；
inputData--发送给 TA 的数据；

inputDataLen--数据长度；
applicationContext--应用上下文，通常在初始化的时候由平台提供给应用；
outputData--从 TA 返回的数据。
返回：int，0 表示成功，其他表示错误。

F.2.6.10 DCASAL_getChipPublicId接口

原型：int DCASAL_getChipPublicId(DCAS_ChipId_t* chipid);
功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，获取 ChipId。
参数：chipid--终端安全芯片唯一 ID。
返回：int，0 表示成功，其他表示错误。

F.2.6.11 DCASAL_sendDescramblingEvent接口

原型：int DCASAL_sendDescramblingEvent(unsigned int casId, DCAS_CASSession_t* casSession, DCAS_CASStatus_t* casStatus);
功能：本方法是 DCAS 框架层调用 DCAS 组件的接口，通过调用这个方法把 DCAS 解扰状态发送给平台显示。
参数：casId--CASModule 管理的 caSystemId；
casSession--从 onSTartDescrambling 获得的 CAS Session；
casStatus--CASStatus 对象。
返回：int，0 表示成功，其他表示错误。

F.2.6.12 DCASAL_setData接口

原型：int DCASAL_setData(int casId, int cmdId, int type, unsigned char* data, int length);
功能：存储接口函数。由DCAS组件实现，供EPG或浏览器浏览器中间件调用。
参数：casId--CAS系统标识ID；
cmdId--命令的唯一标识，已有类型定义见收发数据命令类型；
type--返回数据类型，已有类型定义见收发数据数据类型；
data--返回数据；
length--返回数据长度。
返回：int，0 表示成功，其他表示错误。

F.2.6.13 DCASAL_getDSD接口

原型：int DCASAL_getDSD();
功能：本方法用于 DCAS 应用获得可分离设备（智能卡等）的句柄。
输入：无。
输出：int，表示 DSD 句柄。

F.2.6.14 DCASAL_registerDSDListener接口

原型：void DCASAL_registerDSDListener(unsigned int casId, DSDStatus *dsdStatus);
功能：本方法用于 DCAS 应用注册接收可分离安全设备发送数据的侦听器。
输入：casId--CASModule 管理的 caSystemId；
dsdStatus--DSD 状态变化回调函数。

返回：无。

F.2.6.15 DCASAL_removeDSDListener接口

原型：void DCASAL_removeDSDListener(unsigned int casId);
功能：本方法用于 DCAS 应用来删除已注册的侦听器。
输入：casId--CASModule 管理的 caSystemId。
返回：无。

F.2.6.16 DCASAL_getData接口

原型：int DCASAL_getData(int casId, int cmdId, int& type, unsigned char* data, int& length);
功能：查询数据接口函数。由DCAS组件实现，供EPG或浏览器中间件调用。
参数：casId--CAS系统标识ID;
cmdId--命令的唯一标识;
type--p返回数据类型;
data--返回数据;
length--返回数据长度。
返回：int，0 表示成功，-1 表示失败。

F.2.6.17 DCASAL_getCAInfo接口

原型：void DCASAL_getCAInfo(int casId, int cmdId, unsigned char* paramdata, int paramlength, unsigned char* data, int& length);
功能：查询CA信息函数。由DCAS组件实现，供主应用调用。
参数：casId--CAS系统标识ID;
cmdId--命令的唯一标识;
paramdata--设置的数据;
paramlength--设置的数据长度;
data--设置的数据;
length--设置的数据长度。
返回：无。

F.2.6.18 DCASAL_setCAInfo接口

原型：void DCASAL_setCAInfo(int casId, int cmdId, unsigned char* data, int length);
功能：设置CA信息函数。由DCAS组件实现，供主应用调用。
参数：casId--CAS系统标识ID;
cmdId--命令的唯一标识;
data--设置的数据;
length: 设置的数据长度。
返回：无。

F.3 解扰操作功能模块

F.3.1 概述

本模块定义了解扰操作相关功能接口，见表F.3。

表 F.3　解扰操作功能模块接口

接口	说明
DCASAL_startDescrambling	开始解扰接口。
DCASAL_updateDescrambling	更新解扰接口。
DCASAL_stopDescrambling	停止解扰接口。
DCASAL_notifyFreqChange	频点切换通知。

F.3.2　常量定义

F.3.2.1　KLAD中KEY

原型：#define PARITY_EVEN　0
描述：偶密钥。
原型：#define PARITY_ODD　1
描述：奇密钥。

F.3.2.2　KLAD中算法

原型：#define SCHEME_3DES　0
描述：KLAD使用3DES算法。
原型：#define SCHEME_AES　1
描述：KLAD使用AES算法。

F.3.2.3　通知解扰返回消息

原型：#define KLAD_CW　1
描述：返回的消息是KLAD的CW。
原型：#define CLEAR_CW　2
描述：返回的消息是CLEAR的CW。
原型：#define CA_READY　3
描述：DCAS组件准备好了。
原型：#define CA_NOT_READY　4
描述：DCAS组件尚未准备好。
原型：#define DCAS_SERVER_DEAD　5
描述：DCAS组件异常退出。

F.3.3　事件类型

无。

F.3.4　数据结构

F.3.4.1　PMT表CA信息

原型：typedef struct {
　　int cas_id;

```
    int ecm_pid;
    int emm_pid;
    unsigned char private_data[32];
} DCAS_CAData_t;
```

描述：PMT 表中 CA 信息定义。

成员：

cas_id：CA 的 SystemID；

ecm_pid：ECM 的 PID；

emm_pid：EMM 的 PID；

private_data：定长 32 个字节的私有描述符数据。

F.3.4.2 PMT表ES信息

```
原型：typedef struct {
    int es_pid;
    int es_type;
    DCAS_CAData_t ca_data[8];
} DCAS_ESData_t;
```

描述：PMT 表中 ES 信息定义。

成员：

es_pid：码流中视频、音频或私有描述符的PID；

es_type：码流中视频、音频的类型，私有描述符该字段添0x1fff；

ca_data：定长8个的PMT表中CA信息定义。

F.3.4.3 通知解扰参数

```
原型：typedef struct {
    int demux_id;
    int service_id;
    int tuner_id;
    int tsid;
    int nid;
    int onid;
    int program_num;
    int channel_num;
    int transmitter_scramblingmode;
    unsigned int streamPath[3];
    DCAS_ESData_t es_data[8];
} DCAS_NotifyData_t;
```

描述：通知解扰参数定义。

成员：

demux_id：解复用器ID；

service_id：节目服务ID；

tuner_id：Tuner的ID；

tsid：传输流ID；
nid：网络ID；
onid：原始网络ID；
program_num：节目号；
channel_num：频道号；
transmitter_scramblingmode：码流CW加扰模式；
streamPath：定长3个int类型的安全视频路径参数；
es_data：定长8个的PMT表中ES信息定义。

F.3.4.4 通知解扰返回参数

原型：typedef struct {
    int msg_type;
    int demux_id;
    int vendor_id;
    int data_pid;
    Key level_keys[3];
    int level_keys_len;
    CWKey cw_key;
    int scheme_id;
} DCAS_NotifyRetData_t;
描述：通知解扰返回参数定义。
成员：
msg_type：见定义通知解扰返回消息；
demux_id：解复用器ID；
vendor_id：KLAD分配的厂商的ID；
data_pid：音视频节目的PID；
level_keys：长度3个的KLAD的层级密钥；
level_keys_len：层级密钥的层数；
cw_key：音视频需要CW；
scheme_id：KLAD的加密算法。

F.3.4.5 设置KLAD的KEY参数

原型：typedef struct Key {
    unsigned char value[64];
    unsigned int length;
    bool encrypted;
} DCAS_Key_t;
描述：设置 KLAD 的 KEY 参数定义。
成员：
value：Key的数组；
length：Key的长度；
encrypted：Key是否加密。

F.3.4.6 CW参数

原型：typedef struct CWKey {
    DCAS_Key_t key;
    int parity;
} DCAS_CWKey_t;
描述：CW 的参数定义。
成员：
key：见设置KLAD的KEY参数定义；
parity：奇偶属性。

F.3.4.7 频点切换通知参数

原型：typedef struct {
    int tuner_id;
    int service_id;
    int tsid;
    int nid;
    int onid;
} DCAS_TSData_t;
描述：频点切换通知参数定义。
成员：
tuner_id：Tuner的ID；
service_id：服务ID；
tsid：传输流ID；
nid：网络ID；
onid：原始网络ID。

F.3.5 回调函数定义

F.3.5.1 DCASAL_descramblingNotify回调

原型：typedef void (*DCASAL_descramblingNotify)(void *, void *);
功能：解扰开始、解扰更新和解扰停止的通知回调函数。
参数：void*--DCAS_NotifyRetData_t参数指针；
      void*--播放器函数指针。
返回：无。

F.3.6 接口定义

F.3.6.1 DCASAL_startDescrambling接口

原型：void DCASAL_startDescrambling(void* dcas_notify_data, DCASAL_descramblingNotify cb,
      void* cb_priv);
功能：本方法从 DVBPlayer 调用 DCAS 组件的接口，启动节目解扰。
参数：dcas_notify_data--解扰节目需要的参数，见 DCAS_NotifyData_t；

cb--回调函数；
cb_priv--回调函数参数，对象指针。
返回：无。

F.3.6.2 DCASAL_updateDescrambling接口

原型：void DCASAL_updateDescrambling(void* dcas_notify_data, DCASAL_descramblingNotify cb, void* cb_priv);
功能：本方法从 DVBPlayer 调用 DCAS 组件的接口，更新节目解扰。
参数：dcas_notify_data--解扰节目需要的参数，见 DCAS_NotifyData_t；
cb--回调函数；
cb_priv，回调函数参数，对象指针。
返回：无。

F.3.6.3 DCASAL_stopDescrambling接口

原型：void DCASAL_stopDescrambling(void* dcas_notify_data,DCASAL_descramblingNotify cb, void* cb_priv);
功能：本方法从 DVBPlayer 调用 DCAS 组件的接口，停止节目解扰。
参数：dcas_notify_data--解扰节目需要的参数，见 DCAS_NotifyData_t；
cb--回调函数；
cb_priv，回调函数参数，对象指针。
返回：无。

F.3.6.4 DCASAL_notifyFreqChange接口

原型：void DCASAL_notifyFreqChange(void *ts_data);
功能：本方法从 DVBPlayer 调用 DCAS 组件的接口，频点切换通知。
参数：ts_data--解扰节目需要的参数，见 DCAS_TSData_t。
返回：无。

附 录 G
（规范性附录）
人机交互组件

G.1 概述

本附录定义了人机交互组件对外提供的接口，包括键盘与鼠标消息处理和语音消息处理功能模块对外接口，见表 G.1。

表 G.1 人机交互组件功能模块

模块	说明
键盘与鼠标消息处理功能模块	定义了键盘与鼠标消息处理功能接口。
语音消息处理功能模块	定义了语音消息处理功能接口。

G.2 键盘与鼠标消息处理功能模块

G.2.1 概述

本模块定义了键盘与鼠标消息处理功能接口，见表G.2。

表 G.2 键盘与鼠标消息处理功能模块接口

接口	说明
getDeviceClasses	获取设备类别。
getDeviceIdentifier	获取设备描述信息。
getDeviceControllerNumber	获取设备对应的控制号。
getConfiguration	获取设备对应的配置信息。
getAbsoluteAxisInfo	获取绝对轴信息。
hasRelativeAxis	判断是否有相对轴。
hasInputProperty	判断设备是否具有某属性。
mapKey	将系统层的按键映射到应用层的定义。
mapAxis	将系统层的原始轴信息映射到应用层的定义。
setExcludedDevices	设置需要忽略的设备。
getEvents	获取设备原始事件。
getScanCodeState	获取指定设备的指定扫描码的状态。
getKeyCodeState	获取指定设备的指定按键的状态。
getSwitchState	获取指定开关的状态。
getAbsoluteAxisValue	获取指定设备的指定轴的绝对值。
markSupportedKeyCodes	标记支持的按键。
hasScanCode	获取指定设备是否支持指定扫描码。
hasLed	获取指定设备是否有指定LED。
setLedState	设置指定设备的指定LED的状态。

表 G.2（续）

接口	说明
setFrontPanelString	设置前面板的显示字符。
getVirtualKeyDefinitions	获取指定设备的虚拟按键定义。
getKeyCharacterMap	获取指定设备的按键映射表。
setKeyboardLayoutOverlay	设定指定设备的按键映射表。
vibrate	设置指定设备的震动时间。
cancelVibrate	取消指定设备的震动。
requestReopenDevices	请求重新打开设备。
wake	设备输入唤醒。

G.2.2 常量定义

无。

G.2.3 事件类型

无。

G.2.4 数据结构

G.2.4.1 输入设备类型

原型：enum {

```
    INPUT_DEVICE_CLASS_KEYBOARD      = 0x00000001,
    INPUT_DEVICE_CLASS_ALPHAKEY      = 0x00000002,
    INPUT_DEVICE_CLASS_TOUCH         = 0x00000004,
    INPUT_DEVICE_CLASS_CURSOR        = 0x00000008,
    INPUT_DEVICE_CLASS_TOUCH_MT      = 0x00000010,
    INPUT_DEVICE_CLASS_DPAD          = 0x00000020,
    INPUT_DEVICE_CLASS_GAMEPAD       = 0x00000040,
    INPUT_DEVICE_CLASS_SWITCH        = 0x00000080,
    INPUT_DEVICE_CLASS_JOYSTICK      = 0x00000100,
    INPUT_DEVICE_CLASS_VIBRATOR      = 0x00000200,
    INPUT_DEVICE_CLASS_VIRTUAL       = 0x40000000,
    INPUT_DEVICE_CLASS_EXTERNAL      = 0x80000000,
};
```

描述：人机交互输入设备类型定义。

成员：

INPUT_DEVICE_CLASS_KEYBOARD：键盘或按钮；

INPUT_DEVICE_CLASS_ALPHAKEY：alpha-numeric 键盘（不仅仅是拨号面板）；

INPUT_DEVICE_CLASS_TOUCH：触摸屏或触摸板（单点或多点触摸）；

INPUT_DEVICE_CLASS_CURSOR：轨迹球或鼠标；

INPUT_DEVICE_CLASS_TOUCH_MT：多点触摸屏；

INPUT_DEVICE_CLASS_DPAD：方向板（可能是支持DPAD按键的键盘）；

INPUT_DEVICE_CLASS_GAMEPAD：游戏板（可能是支持按钮的键盘）；
INPUT_DEVICE_CLASS_SWITCH：开关；
INPUT_DEVICE_CLASS_JOYSTICK：操纵杆（可能是带操纵杆的游戏板）；
INPUT_DEVICE_CLASS_VIBRATOR：振动器；
INPUT_DEVICE_CLASS_VIRTUAL：虚拟设备；
INPUT_DEVICE_CLASS_EXTERNAL：外部设备。

G.2.4.2 原始事件结构

原型：struct RawEvent {
    nsecs_t when;
    int32_t deviceId;
    int32_t type;
    int32_t code;
    int32_t value;
};
描述：输入设备产生的底层事件。
成员：
when：事件时间；
deviceId：输入设备ID；
type：事件类型；
code：事件码；
value：事件值。

G.2.4.3 事件状态类型

原型：enum {
    AKEY_STATE_UNKNOWN = -1,
    AKEY_STATE_UP = 0,
    AKEY_STATE_DOWN = 1,
    AKEY_STATE_VIRTUAL = 2
};
描述：事件（按键、开关等）的状态。
成员：
AKEY_STATE_UNKNOWN：状态未知或按键等不支持；
AKEY_STATE_UP：键抬起；
AKEY_STATE_DOWN：键按下；
AKEY_STATE_VIRTUAL：键按下（系统模拟的虚拟事件）。

G.2.5 回调函数定义

无。

G.2.6 接口定义

G.2.6.1 getDeviceClasses接口

原型：uint32_t getDeviceClasses(int32_t deviceId);
功能：获取设备类别。
参数：deviceId--设备 ID。
返回：uint32_t，设备类别，参见 G.2.4.1 输入设备类型。

G.2.6.2 getDeviceIdentifier接口

原型：InputDeviceIdentifier getDeviceIdentifier(int32_t deviceId);
功能：获取设备描述信息。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）。
返回：InputDeviceIdentifier，设备描述信息。

G.2.6.3 getDeviceControllerNumber接口

原型：int32_t getDeviceControllerNumber(int32_t deviceId);
功能：获取设备对应的控制号。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）。
返回：int32_t，设备对应的控制号。

G.2.6.4 getConfiguration接口

原型：void getConfiguration(int32_t deviceId, PropertyMap* outConfiguration);
功能：获取设备对应的配置信息。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
outConfiguration--输出参数，设备配置信息。
返回：无。

G.2.6.5 getAbsoluteAxisInfo接口

原型：status_t getAbsoluteAxisInfo(int32_t deviceId, int axis, RawAbsoluteAxisInfo* outAxisInfo);
功能：获取绝对轴信息。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
axis--轴 ID（0x0～0x3f）；
outAxisInfo--结构体，轴信息。
返回：status_t，轴信息获取状态，0 表示成功，小于 0 表示失败。

G.2.6.6 hasRelativeAxis接口

原型：bool hasRelativeAxis(int32_t deviceId, int axis);
功能：判断是否有相对轴。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
axis--轴 ID（0x0～0x3f）。
返回：bool，有相对轴返回 true；无相对轴返回 false。

G.2.6.7 hasInputProperty接口

原型：bool hasInputProperty(int32_t deviceId, int property);

功能：判断设备是否具有某属性。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
property--属性 ID（0x0～0x1f）。
返回：bool，有此属性返回 true；无此属性返回 false。

G.2.6.8 mapKey接口

原型：status_t mapKey(int32_t deviceId, int32_t scanCode, int32_t usageCode,
int32_t* outKeycode, uint32_t* outFlags);
功能：将系统层的按键映射到应用层的定义。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
scanCode--按键扫描码；
usageCode--按键使用码；
outKeycode--映射后的按键值；
outFlags--映射后的按键标记。
返回：status_t，键值映射状态，0 表示成功，小于 0 表示失败。

G.2.6.9 mapAxis接口

原型：status_t mapAxis(int32_t deviceId, int32_t scanCode, AxisInfo* outAxisInfo);
功能：将系统层的原始轴信息映射到应用层的定义。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
scanCode--扫描码；
outAxisInfo--映射后的轴信息。
返回：status_t，轴信息映射状态，0 表示成功，小于 0 表示失败。

G.2.6.10 setExcludedDevices接口

原型：void setExcludedDevices(const Vector<String8>& devices);
功能：设置需要忽略的设备。
参数：devices--需忽略的设备名称。
返回：无。

G.2.6.11 getEvents接口

原型：size_t getEvents(int timeoutMillis, RawEvent* buffer, size_t bufferSize);
功能：获取设备原始事件。
参数：timeoutMillis--超时时间；
buffer-- 结构体指针，存储获取到的原始事件，参见原始事件结构；
bufferSize--buffer 的大小。
返回：size_t，获取到的事件数量。

G.2.6.12 getScanCodeState接口

原型：int32_t getScanCodeState(int32_t deviceId, int32_t scanCode);
功能：获取指定设备的指定扫描码的状态。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；

scanCode--扫描码。

返回：int32_t，扫描码状态，参见事件状态类型。

G.2.6.13 getKeyCodeState接口

原型：int32_t getKeyCodeState(int32_t deviceId, int32_t keyCode);

功能：获取指定设备的指定按键的状态。

参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；

keyCode--键值。

返回：int32_t，获取到的按键状态，参见事件状态类型。

G.2.6.14 getSwitchState接口

原型：int32_t getSwitchState(int32_t deviceId, int32_t sw);

功能：获取指定开关的状态。

参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；

sw--开关 ID。

返回：int32_t，开关状态，参见事件状态类型。

G.2.6.15 getAbsoluteAxisValue接口

原型：status_t getAbsoluteAxisValue(int32_t deviceId,int32_t axis,int32_t* outValue);

功能：获取指定设备的指定轴的绝对值。

参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；

axis--轴的 ID（0x0～0x3f）；

outValue--输出参数，指定轴的值。

返回：status_t，获取状态，0 表示成功，小于 0 表示失败。

G.2.6.16 markSupportedKeyCodes接口

原型：bool markSupportedKeyCodes(int32_t deviceId, size_t numCodes, const int32_t* keyCodes, uint8_t* outFlags);

功能：标记支持的按键。

参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；

numCodes--码值的个数；

keyCodes--码值信息；

outFlags--码值的标记。

返回：bool，表示是否成功。

G.2.6.17 hasScanCode接口

原型：bool hasScanCode(int32_t deviceId, int32_t scanCode);

功能：获取指定设备是否支持指定扫描码。

参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；

scanCode--扫描码。

返回：bool，扫描码是否支持。

G.2.6.18 hasLed接口

原型：bool hasLed(int32_t deviceId, int32_t led);
功能：获取指定设备是否有指定 LED。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
led--LED 的 ID。
返回：bool，指定设备是否有 LED。

G.2.6.19 setLedState接口

原型：void setLedState(int32_t deviceId, int32_t led, bool on);
功能：设置指定设备的指定 LED 的状态。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
led--LED 的 ID;
on--要设置的 LED 状态。
返回：无。

G.2.6.20 setFrontPanelString接口

原型：void setFrontPanelString(int32_t deviceId, String8 content);
功能：设置前面板的显示字符。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
content --要显示的字符。
返回：无。

G.2.6.21 getVirtualKeyDefinitions接口

原型：void getVirtualKeyDefinitions(int32_t deviceId,
Vector<VirtualKeyDefinition>& outVirtualKeys);
功能：获取指定设备的虚拟按键定义。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
outVirtualKeys--输出参数，虚拟按键定义。
返回：无。

G.2.6.22 getKeyCharacterMap接口

原型：sp<KeyCharacterMap> getKeyCharacterMap(int32_t deviceId);
功能：获取指定设备的按键映射表。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）。
返回：sp<KeyCharacterMap>，指定设备按键映射表。

G.2.6.23 setKeyboardLayoutOverlay接口

原型：bool setKeyboardLayoutOverlay(int32_t deviceId, const sp<KeyCharacterMap>& map);
功能：设定指定设备的按键映射表。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
map--按键映射表。
返回：bool，设定映射表是否成功。

G.2.6.24 vibrate接口

原型：void vibrate(int32_t deviceId, nsecs_t duration);
功能：设置指定设备的震动时间。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）；
duration--震动时间。
返回：无。

G.2.6.25 cancelVibrate接口

原型：void cancelVibrate(int32_t deviceId);
功能：取消指定设备的震动。
参数：deviceId--设备 ID（-1:虚拟键盘，0:内建键盘，正数：其它设备）。
返回：无。

G.2.6.26 requestReopenDevices接口

原型：void requestReopenDevices();
功能：请求重新打开设备。
参数：无。
返回：无。

G.2.6.27 wake接口

原型：void wake();
功能：设备输入唤醒。
参数：无。
返回：无。

G.3 语音消息处理功能模块

G.3.1 概述

本模块定义了语音消息处理的功能接口，见表G.3。

表 G.3 语音消息处理功能模块接口

接口	说明
hci_vpInstance_create	创建识别实例。
hci_vpRecognize_start	启动语音识别输入。
hci_vpRecognize_stop	停止语音识别输入。
hci_vpRecognize_cancel	取消语音识别输入。
hci_vpRecognize_destroy	销毁语音识别引擎实例。

G.3.2 常量定义

无。

G.3.3 事件类型

无。

G.3.4 数据结构

G.3.4.1 语音识别引擎厂家

原型：enum HCI_VP_VENDOR_e {
    HCI_VP_BAIDU = 0x00,
    HCI_VP_IFLYTEK = 0x01,
    HCI_VP_AISPEECH = 0x02,
    HCI_VP_UNDEFINED = 0x03
};
描述：标识不同的语音识别引擎厂家。
成员：
HCI_VP_BAIDU：百度引擎；
HCI_VP_IFLYTEK：科大讯飞引擎；
HCI_VP_AISPEECH：思必驰引擎；
HCI_VP_UNDEFINED：未定义。

G.3.4.2 语音识别结果

原型：enum HCI_VP_RUSULT_e {
    HCI_VP_OK = 0,
    HCI_VP_TOO_SHORT = -1,
    HCI_VP_ NOT_MANDARIN = -2,
    HCI_VP_TOO_FAST = -3,
    HCI_VP_OTHER = -4
};
描述：语音识别结果返回码定义。
成员：
HCI_VP_OK：识别正常；
HCI_VP_TOO_SHORT：音频太短；
HCI_VP_ NOT_MANDARIN：非普通话；
HCI_VP_TOO_FAST：语速太快；
HCI_VP_OTHER：其它错误。

G.3.5 回调函数定义

G.3.5.1 hci_vp_callback回调

原型：typedef void(*hci_vp_callback)(int handle, HCI_VP_RUSULT_e code, char *results);
功能：语音识别结果输出，供语音引擎回调。
参数：handle--语音识别实例句柄；
code--返回码，见语音识别结果。

results--识别结果。

返回：无。

G.3.5.2 hci_vpData_inject回调

原型：typedef void(*hci_vpData_inject) (int len,char *data)

功能：语音数据注入，供语音引擎回调。

参数：len--音频数据长度；

data--音频数据。

返回：无。

G.3.6 接口定义

G.3.6.1 hci_vpInstance_create接口

原型：int hci_vpInstance_create (HCI_VP_VENDOR_e vendor, hci_vp_callback func);

功能：创建识别实例。

参数：vendor--语音引擎厂商，见语音识别引擎厂家。

func--语音识别结果回调。

返回：int，实例句柄，-1 表示失败。

G.3.6.2 hci_vpRecognize_start接口

原型：int hci_vpRecognize_start (int handle, hci_vpData_inject func);

功能：启动语音识别输入。接口内部开始读取音频数据。

参数：handle--语音识别实例句柄；

func--音频数据读取函数入口。

返回：int，0 表示成功，其它表示失败。

G.3.6.3 hci_vpRecognize_stop接口

原型：int hci_vpRecognize_stop (int handle);

功能：停止语音识别输入。接口内部停止读取音频数据，并进行语音识别。

参数：handle--语音识别实例句柄。

返回：int，0 表示成功，其它表示失败。

G.3.6.4 hci_vpRecognize_cancel接口

原型：void hci_vpRecognize_cancel (int handle);

功能：取消语音识别输入。

参数：handle--语音识别实例句柄。

返回：无。

G.3.6.5 hci_vpRecognize_destroy接口

原型：int hci_vpRecognize_destroy (int handle);

功能：销毁语音识别引擎实例。

参数：handle--语音识别引擎实例。

返回：int，0 表示成功，其它表示失败。

附 录 H
（规范性附录）
多屏互动组件

H.1 概述

本附录定义了多屏互动组件对外提供的接口，包括设备发现及连接功能模块和跨屏 UI 操控功能模块对外接口,见表 H.1。

表 H.1 多屏互动组件功能模块

模块	说明
设备发现及连接功能模块	定义了多屏互动组件设备发现功能模块的接口。
跨屏 UI 操控功能模块	定义了多屏互动组件跨屏 UI 操控的接口。

H.2 设备发现及连接功能模块

H.2.1 概述

本模块定义了多屏互动组件设备发现功能模块的接口，客户端模块能够发现并且连接处于相同局域网中的服务端模块，是多屏互动组件当中重要的组成模块,见表H.2。

表 H.2 设备发现及连接功能模块接口

接口	说明
startMultiScreenClient	启动发起端。
stopMultiScreenClient	与接收端断开连接。
findSPs	发现多屏互动组件服务。
connect	初始化多屏互动组件服务，连接接收端。
setCallBack	设置组件远程接口的回调。

H.2.2 常量定义

H.2.2.1 多屏互动组件错误码

```
原型：enum {
    OK                = 0,
    NO_ERROR          = 0,
    UNKNOWN_ERROR       = (-2147483647-1),
    NO_MEMORY           = -ENOMEM,
    INVALID_OPERATION   = -ENOSYS,
    BAD_VALUE           = -EINVAL,
    BAD_TYPE            = (UNKNOWN_ERROR + 1),
    NAME_NOT_FOUND      = -ENOENT,
    PERMISSION_DENIED   = -EPERM,
```

```
    NO_INIT             = -ENODEV,
    ALREADY_EXISTS      = -EEXIST,
    DEAD_OBJECT         = -EPIPE,
    FAILED_TRANSACTION  = (UNKNOWN_ERROR + 2),
    JPARKS_BROKE_IT     = -EPIPE,
#if !defined(HAVE_MS_C_RUNTIME)
    BAD_INDEX           = -EOVERFLOW,
    NOT_ENOUGH_DATA     = -ENODATA,
    WOULD_BLOCK         = -EWOULDBLOCK,
    TIMED_OUT           = -ETIMEDOUT,
    UNKNOWN_TRANSACTION = -EBADMSG,
#else
    BAD_INDEX           = -E2BIG,
    NOT_ENOUGH_DATA     = (UNKNOWN_ERROR + 3),
    WOULD_BLOCK         = (UNKNOWN_ERROR + 4),
    TIMED_OUT           = (UNKNOWN_ERROR + 5),
    UNKNOWN_TRANSACTION = (UNKNOWN_ERROR + 6),
#endif
    FDS_NOT_ALLOWED     = (UNKNOWN_ERROR + 7),
};
```

描述：错误码定义。

成员：

OK：状态正常；

NO_ERROR：无错误；

UNKNOWN_ERROR：INT32_MIN 值；

NO_MEMORY：缺少内存；

INVALID_OPERATION：无效操作；

BAD_VALUE：错误值；

BAD_TYPE：错误类型；

NAME_NOT_FOUND：找不到命名；

PERMISSION_DENIED：权限拒绝；

NO_INIT：未初始化；

ALREADY_EXISTS：已存在；

DEAD_OBJECT：对象已销毁；

FAILED_TRANSACTION：业务失败；

JPARKS_BROKE_IT：通道已损坏；

BAD_INDEX：错误索引；

NOT_ENOUGH_DATA：缺少足够数据；

WOULD_BLOCK ：将产生阻塞；

TIMED_OUT：超时；

UNKNOWN_TRANSACTION：未知业务；

FDS_NOT_ALLOWED：不允许的FDS。

H.2.2.2 远程接口索引枚举

原型： enum {
START_MULTISCREENSERVER = 0x0,
START_MULTISCREENSERVER,
START_MULTISCREENCLIENT,
STOP_MULTISCREENCLIENT,
FIND_SPS,
CONNECT,
SET_CALLBACK,
QUERY_INFO,
EXEC_CMD,
INPUT_KEYCODE,
NOTIFY_ALL_REMOTE
};
描述：多屏互动组件远程接口索引。
成员：
START_MULTISCREENSERVER：启动多屏互动组件服务端远程接口；
STOP_MULTISCREENSERVER：关闭多屏服务服务端远程接口；
START_MULTISCREENCLIENT：启动多屏互动组件客户端远程接口；
STOP_MULTISCREENCLIENT：关闭多屏互动组件客户端远程接口；
FIND_SPS：发现局域网内的多屏互动组件服务端远程接口；
CONNECT：连接局域网内的多屏互动组件服务端远程接口；
SET_CALLBACK：设置多屏互动组件回调远程接口；
QUERY_INFO：请求多屏互动组件服务端信息的远程接口；
EXEC_CMD：多屏互动组件客户端发送执行指令给到多屏互动服务端的远程接口；
INPUT_KEYCODE：多屏互动组件客户端发送按键指令给到多屏互动服务端的远程接口；
NOTIFY_ALL_REMOTE：多屏互动组件服务端发送通知给到多屏互动客户端的远程接口。

H.2.3 事件类型

无。

H.2.4 数据结构

无。

H.2.5 回调函数定义

H.2.5.1 onSpFounded回调

原型：status_t onSpFounded(const char* spName, const char* spDeviceType,
const char* spServiceInfo, const char* spVersion, const char* ipaddress,
int port, const char* hostname);
功能：远程接口回调，通知使用多屏组件的远端程序发现到了局域网内其他多屏互动组件服务的设备。
参数：spName--输出参数，局域网中的多屏互动组件设备服务名称；

spDeviceType--输出参数，局域网中的多屏互动组件设备类型；
spServiceInfo--输出参数，局域网中的多屏互动组件设备服务信息；
spVersion--输出参数，局域网中的多屏互动组件设备服务版本；
ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.2.5.2 onConnected回调

原型：status_t onConnected(const char* spName, const char* spDeviceType,
const char* spServiceInfo, const char* spVersion, const char* ipaddress,
int port, const char* hostname);

功能：远程接口回调，通知使用多屏组件的远端程序连接上了局域网内其他多屏互动组件服务的设备。

参数：spName--输出参数，局域网中的多屏互动组件设备服务名称；
spDeviceType--输出参数，局域网中的多屏互动组件设备类型；
spServiceInfo--输出参数，局域网中的多屏互动组件设备服务信息；
spVersion--输出参数，局域网中的多屏互动组件设备服务版本；
ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.2.5.3 onConnectRefused回调

原型：status_t onConnectRefused(const char* spName, const char* spDeviceType,
const char* spServiceInfo, const char* spVersion, const char* ipaddress,
int port, const char* hostname);

功能：远程接口回调，通知使用多屏组件的远端程序被拒绝连接上局域网内其他多屏互动组件服务的设备。

参数：spName--输出参数，局域网中的多屏互动组件设备服务名称；
spDeviceType--输出参数，局域网中的多屏互动组件设备类型；
spServiceInfo--输出参数，局域网中的多屏互动组件设备服务信息；
spVersion--输出参数，局域网中的多屏互动组件设备服务版本；
ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.2.5.4 onDisconnected回调

原型：status_t onDisconnected(const char* spName, const char* spDeviceType,
const char* spServiceInfo, const char* spVersion, const char* ipaddress,
int port, const char* hostname);

功能：远程接口回调，通知使用多屏组件的远端程序与局域网内其他多屏互动组件服务的设备断开连

接。

参数：spName--输出参数，局域网中的多屏互动组件设备服务名称；
spDeviceType--输出参数，局域网中的多屏互动组件设备类型；
spServiceInfo--输出参数，局域网中的多屏互动组件设备服务信息；
spVersion--输出参数，局域网中的多屏互动组件设备服务版本；
ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port-输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.2.5.5 onServiceActived回调

原型：status_t onServiceActived(const char* ipaddress, int port, const char* hostname);

功能：远程接口回调，通知使用多屏组件的远端程序被局域网内其他多屏互动组件连接上了服务。

参数：ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.2.5.6 onServiceDeactived回调

原型：status_t onServiceDeactived(const char* ipaddress, int port, const char* hostname);

功能：远程接口回调，通知使用多屏组件的远端程序被局域网内其他多屏互动组件断开了服务。

参数：ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.2.6 接口定义

H.2.6.1 startMultiScreenClient接口

原型：status_t startMultiScreenClient(const char* clientName);

功能：启动发起端。

参数：clientName--输入参数，客户端模块的命名。

返回：status_t，远程接口调用成功返回 0，否则返回错误码。

H.2.6.2 stopMultiScreenClient 接口

原型：status_t stopMultiScreenClient();

功能：与接收端断开链接。

参数：无。

返回：status_t，远程接口调用成功返回 0，否则返回错误码。

H.2.6.3 findSPs接口

原型：status_t findSPs();

功能：发现局域网中多屏互动组件服务。

参数：无。

返回：int，正常则返回NO_ERROR，否则返回错误码。

H.2.6.4 connect接口

原型：status_t connect(const char* spName, const char* spDeviceType, const char* spServiceInfo, const char* spVersion, const char* ipaddress, int port, const char* hostname);

功能：初始化多屏互动组件提供的服务，连接接收端。

参数：spName--输入参数，局域网中的多屏互动组件设备服务名称；

spDeviceType--输入参数，局域网中的多屏互动组件设备类型；

spServiceInfo--输入参数，局域网中的多屏互动组件设备服务信息；

spVersion--输入参数，局域网中的多屏互动组件设备服务版本；

ipaddress--输入参数，局域网中的多屏互动组件设备 ip 地址；

port--输出参数，局域网中的多屏互动组件设备端口号；

hostname--输入参数，局域网中的多屏互动组件设备主机名。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.2.6.5 setCallBack接口

原型：status_t setCallBack(const sp<IMultiScreenCallBack>& multiScreenCallback);

功能：设置多屏互动组件远程接口的回调，使服务端对客户端功能的响应回调。

参数：multiScreenCallback--输入参数，客户端 IMultiScreenCallBack 远程回调接口。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3 跨屏UI操控功能模块

H.3.1 概述

本模块定义了多屏互动组件跨屏UI操控的接口，多屏互动组件客户端能够使用这些接口对局域网内的其他设备发送操控指令，见表H.3。

表 H.3 跨屏 UI 操控功能模块接口

接口	说明
queryInfo	向接收端发送的请求指令。
execCmd	向接收端发送的执行指令请求。
inputKeyCode	向接收端发送的按键事件指令请求。
notifyAllTarget	向已连接的组件设备发送通知。

H.3.2 常量定义

无。

H.3.3 事件类型

无。

H.3.4 数据结构

无。

H.3.5 回调函数定义

H.3.5.1 onQueryInfo回调

原型：status_t onQueryInfo(const char* ipaddress, int port, const char* hostname,
const char* id, const char* attribute, const char* param,
QUERYFUNC queryResult);

功能：远程接口回调，通知远程接口接收到了局域网中其他多屏互动组件设备发过来的请求指令。

参数：ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名；
id--输出参数，请求指令的 id；
attribute--输出参数，请求指令属性名；
param--输出参数，请求指令附带的参数；
queryResult--输入参数，函数指针，用于传递请求结果。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3.5.2 onQueryResponse回调

原型：status_t onQueryResponse(const char* ipaddress, int port, const char* hostname,
const char* id, const char* command, const char* params);

功能：远程接口回调，通知远程接口接收到了局域网中其他多屏互动组件设备发过来的请求指令的相关结果。

参数：ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名；
id--输出参数，请求指令的 id；
attribute--输出参数，请求指令属性名；
param--输出参数，请求指令附带的参数。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3.5.3 onExecute回调

原型：status_t onExecute(const char* ipaddress, int port, const char* hostname,
const char* command, const char* params);

功能：远程接口回调，通知远程接口接收到了局域网中其他多屏互动组件设备发过来的执行指令请求。

参数：ipaddress-- 输出参数，局域网中的多屏互动组件设备 ip 地址；
port-- 输出参数，局域网中的多屏互动组件设备端口号；
hostname-- 输出参数，局域网中的多屏互动组件设备主机名；
command-- 输出参数，请求的执行指令；
params-- 输出参数，请求指令附带的参数。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3.5.4 onInputKeyCode回调

原型：status_t onInputKeyCode(const char* ipaddress, int port, const char* hostname, const char* action, const char* params);

功能：远程接口回调，通知远程接口接收到了局域网中其他多屏互动组件设备发过来的按键事件指令请求。

参数：ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名；
action--输出参数，需要虚拟的按键事件指令；
params--输出参数，按键事件指令附带的参数。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3.5.5 onNotify回调

原型：status_t onNotify(const char* ipaddress, int port, const char* hostname, const char* command, const char* params);

功能：远程接口回调，通知远程接口接收到了局域网中其他多屏互动组件设备发过来的通知事件。

参数：ipaddress--输出参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输出参数，局域网中的多屏互动组件设备主机名；
command--输出参数，通知的指令；
params--输出参数，附带的参数。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3.6 接口定义

H.3.6.1 queryInfo接口

原型：status_t queryInfo(const char* ipaddress, int port, const char* hostname, const char* id, const char* attribute, const char* param);

功能：局域网中多屏互动组件设备发起端向接收端发送的请求指令。

参数：ipaddress--输入参数，局域网中的多屏互动组件设备 ip 地址；
port--输出参数，局域网中的多屏互动组件设备端口号；
hostname--输入参数，局域网中的多屏互动组件设备主机名；
id--输入参数，请求指令的 id；
attribute--输入参数，请求指令属性名；
param--输入参数，请求指令附带的参数。

返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3.6.2 execCmd接口

原型：status_t execCmd(const char* ipaddress, int port, const char* hostname, const char* command, const char* params);

功能：局域网中多屏互动组件设备发起端向接收端发送的执行指令请求。

参数：ipaddress--输入参数，局域网中的多屏互动组件设备 ip 地址；

port--输入参数，局域网中的多屏互动组件设备端口号；
hostname--输入参数，局域网中的多屏互动组件设备主机名；
command--输入参数，请求的执行指令；
params--输入参数，请求指令附带的参数。
返回：status_t，远程接口调用成功返回0，否则返回错误码。

H.3.6.3 inputKeyCode接口

原型：status_t inputKeyCode(const char* ipaddress, int port, const char* hostname, const char* action, const char* params);
功能：局域网中多屏互动组件设备发起端向接收端发送按键事件指令请求。
参数：ipaddress--输入参数，局域网中的多屏互动组件设备 ip 地址；
port--输入参数，局域网中的多屏互动组件设备端口号；
hostname--输入参数，局域网中的多屏互动组件设备主机名；
action--输入参数，需要虚拟的按键事件指令；
params--输入参数，按键事件指令附带的参数。
返回： status_t，远程接口调用成功返回0，否则返回错误码。

H.3.6.4 notifyAllTarget接口

原型：status_t notifyAllTarget(const char* cmd, const char* param);
功能：向局域网内的已经连接上的所有具有多屏互动组件的设备发送通知。
参数：cmd--输入参数，通知的指令；
param--输入参数，附带的参数。
返回：status_t，远程接口调用成功返回 0，否则返回错误码。

附 录 I
（规范性附录）
广播信息服务组件

I.1 概述

本附录定义了广播信息服务组件对外提供的接口，包括广播信息服务业务监测、OSD 更新、广告更新、应急广播监测和信息服务等功能模块的对外接口，见表 I.1。

表 I.1 广播信息服务组件功能模块

模块	说明
广播信息服务业务监测功能模块	定义了广播信息服务组件的启动、停止、注册消息等功能接口。
OSD 更新功能模块	定义 OSD 文本内容更新、获取等功能接口。
广告更新功能模块	定义了开机广告接收、广告信息获取、实时广告接收等接口。
应急广播监测功能模块	定义了应急广播的接口及消息。
信息服务功能模块	定义了信息服务相关的接口。

I.2 广播信息服务业务监测模块

I.2.1 概述

本模块定义了广播信息服务组件的启动、停止、注册消息等功能接口，见表I.2。

表 I.2 广播信息服务业务监测功能模块接口

接口	说明
startServer	启动广播信息服务组件监控。
stopServer	停止广播信息服务业务的监控。
setTunerIdToDth	设置高频头标识到广播信息服务组件。
setListener	注册监听广播信息服务组件消息。
restoreFactorySettings	恢复出厂设置。
SaveNITServiceUpdateVersion	对业务更新描述符的版本号进行保存。
DCASM_sendDataToDTH	DCAS 组件将数据传输给广播信息服务组件。

I.2.2 常量定义

I.2.2.1 DCAS组件收发数据命令

原型：#define ID_SEND_CMD_TO_STB　0x10000001
描述：发送该命令给广播信息服务组件。
原型：#define ID_SEND_DATA_Bouquent_ID　0x10000002
描述：发送该bouqueid给广播信息服务组件。
原型：#define ID_SEND_DATA_ZIP_CODE　0x10000003
描述：区域码。

原型：#define ID_SEND_DATA_PERSONAL_BITS　0x10000004
描述：特征。
原型：#define ID_SEND_DATA_CAS_VENDOR_ID　0x00000005
描述：CAS的Vendor的ID。
原型：#define ID_SEND_DATA_HSM_POSITION_X　0x00000006
描述：HSM的位置X坐标。
原型：#define ID_SEND_DATA_HSM_POSITION_Y　0x00000007
描述：HSM的位置Y坐标。
原型：#define ID_SEND_DATA_CAS_VERSION　0x00000008
描述：CAS的版本。
原型：#define ID_SEND_DATA_TO_HEADEND　0x00000009
描述：发送数据给头端。
原型：#define ID_SEND_DATA_CHIP_ID　0x0000000A
描述：芯片的ID。
原型：#define ID_SEND_DATA_HSM_ID　0x0000000B
描述：HSM的ID。

I.2.2.2　DCAS组件收发数据类型

原型：#define TYPE_INT32　0x01
描述：整数类型。
原型：#define TYPE_INT8ARRAY　0x02
描述：整形数组类型。
原型：#define TYPE_STRING　0x03
描述：字符串类型。

I.2.3　事件类型

无。

I.2.4　数据结构

I.2.4.1　广播信息服务组件事件类型

```
原型：typedef enum _DTH_EVENT_CB_E {
    DTH_EVENT_EMBD_TRIG = 0,
    DTH_EVENT_EMBD_CANCEL,
    DTH_EVENT_OSD_UPDATE,
    DTH_EVENT_SERVICE_UPDATE,
    DTH_EVENT_RESET_DATA,
    DTH_EVENT_BOUQUET_ID_UPDATE,
    DTH_EVENT_UPGRADE_TRIG,
    DTH_EVENT_FINGERPRINT_TRIG,
    DTH_EVENT_DCAS_OSD_TRIG,
    DTH_EVENT_GPRS_STATUS,
```

```
    DTH_EVENT_GPRS_SEND_STATUS,
    DTH_EVENT_GPRS_BASE_STATION,
    DTH_EVENT_BUTT
} DTH_EVENT_CB_E;
```

描述：广播信息服务组件事件类型。

成员：

DTH_EVENT_EMBD_TRIG：应急广播触发；

DTH_EVENT_EMBD_CANCEL：应急广播取消；

DTH_EVENT_OSD_UPDATE：OSD文本更新消息；

DTH_EVENT_SERVICE_UPDATE：业务更新描述符更新消息；

DTH_EVENT_RESET_DATA：擦除数据描述符更新消息；

DTH_EVENT_BOUQUET_ID_UPDATE：bouquetId更新消息；

DTH_EVENT_UPGRADE_TRIG：软件升级触发消息；

DTH_EVENT_FINGERPRINT_TRIG：指纹触发消息；

DTH_EVENT_DCAS_OSD_TRIG：DCAS的OSD文本更新触发消息；

DTH_EVENT_GPRS_STATUS：获取GPRS状态消息（异步用到）；

DTH_EVENT_GPRS_SEND_STATUS：GPRS发送数据状态消息（异步用到）；

DTH_EVENT_GPRS_BASE_STATION：GPRS获取基站消息（异步用到）；

DTH_EVENT_BUTT：保留字段。

I.2.4.2 节目更新描述符

```
原型：typedef struct _Notify_ServiceUpdate_Info {
    HI_S32 DeliverySystemType;
    HI_U16 networkId;
    HI_U8  forceFlag;
} Notify_ServiceUpdate_Info;
```

描述：节目更新描述符结构。

成员：

DeliverySystemType：区分当前上报的是哪种系统类型(10表示卫星，12表示地面)；

networkId：用来标识当前属于哪个网络；

forceFlag：用于确定业务更新是否为强制更新(1表示强制更新，0表示非强制更新)。

I.2.5 回调函数定义

I.2.5.1 notify回调

```
原型：HI_VOID notify(DTH_EVENT_CB_E enMsgType, HI_VOID* pvMsgData);
```

功能：业务消息回调。

参数：enMsgType--消息类型，见DTH_EVENT_CB_E。

pvMsgData--数据指针，根据enMsgType类型转换到相应数据。

- DTH_EVENT_EMBD_TRIG：应急广播触发，参数HI_VOID*为DTH_EMBD_Data_t*，即返回的是应急广播的节目描述。
- DTH_EVENT_EMBD_CANCE：应急广播取消，参数HI_VOID*为DTH_EMBD_Data_t*，即返回的是应急广播的节目描述。

- DTH_EVENT_OSD_UPDATE：OSD文本更新消息，参数HI_VOID*为NULL。
- DTH_EVENT_SERVICE_UPDATE：业务更新描述符更新消息，参数HI_VOID*为Notify_ServiceUpdate_Info*，即返回的是节目更新描述符的类型。
- DTH_EVENT_RESET_DATA：擦除数据描述符更新消息，参数HI_VOID*为HI_U8*，值为1标识擦除数据，值为0标识恢复数据。
- DTH_EVENT_BOUQUET_ID_UPDATE:bouquetId更新消息,参数HI_VOID*为HI_U16*，返回的是更新后的bouquetId值。
- DTH_EVENT_UPGRADE_TRIG：软件升级触发消息，参数HI_VOID*为HI_U8*，值为0标识强制升级，值为1标识非强制升级。
- DTH_EVENT_FINGERPRINT_TRIG：指纹触发消息，参数HI_VOID*为DTH_FingerPrint_t*，即返回的是指纹信息描述。
- DTH_EVENT_DCAS_OSD_TRIG：DCAS的OSD文本更新触发消息，参数HI_VOID*为DTH_DcasOsdText_t*，即返回的是osd信息描述。
- DTH_EVENT_GPRS_STATUS：获取GPRS状态消息，参数HI_VOID*为Notify_Gprs_Status*，即返回的是gprs状态信息。
- DTH_EVENT_GPRS_SEND_STATUS：GPRS发送数据状态消息，参数HI_VOID*为Notify_Gprs_SendData*，即返回的是gprs发送状态信息。
- DTH_EVENT_GPRS_BASE_STATION：GPRS获取基站消息，参数HI_VOID*为Notify_Station_Info*，即返回的是gprs获取到的基站信息。

返回：无。

I.2.6 接口定义

I.2.6.1 startService接口

原型：HI_S32 startServer();
功能：启动广播信息服务组件服务，对NIT及BAT进行监控。
参数：无。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.2.6.2 stopService接口

原型：HI_S32 stopServer();
功能：停止广播信息服务组件服务，停止对NIT及BAT的监控。
参数：无。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.2.6.3 setTunerIdToDth接口

原型：HI_S32 setTunerIdToDth(HI_U32 tunerId);
功能：设置当前需要接收业务的tuner Id类型到广播信息服务组件。
参数：tunerId--输入参数，Tuner Id值。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.2.6.4 setListener接口

原型：HI_S32 setListener(DTHServiceListener* pListener);
功能：对广播信息服务组件回报消息的处理进行注册。
参数：pListener--输入参数，该指针指向接口类DTHServiceListener的派生类，DTHServiceListener的派生类需要重写notify方法。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.2.6.5 restoreFactorySettings接口

原型：HI_S32 restoreFactorySettings();
功能：对广播信息服务组件保存的数据进行恢复出厂设置。
参数：无。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.2.6.6 SaveNITServiceUpdateVersion接口

原型：HI_S32 SaveNITServiceUpdateVersion(HI_S32 deliveryType, HI_U16 networkId, HI_U8 version = 0xff);
功能：对业务更新描述符的版本号进行保存。
参数：deliveryType--输入参数,标识当前需要保存该标志的系统类型。10标识卫星，12标识地面。
networkId—输入参数，标识当前系统的network id。
version--输入参数，当值为0xff时，以接收到的版本号为主保存，非0xff时以输入的版本号为主。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.2.6.7 DCASM_sendDataToDTH接口

原型：HI_S32 DCASM_sendDataToDTH(HI_S32 casid, HI_U32 cmdId, HI_S32 type, HI_U8 *data, HI_S32 length);
功能：由DCAS组件调用，将数据传输给广播信息服务组件。
参数：casid--输入参数，CAS系统的唯一标识；
cmdId--输入参数，命令的唯一标识；
type--输入参数，data所指数据的类型，便于转换data存储的数据；
data--输入参数，指向实际数据的指针；
length--输入参数，data所指数据的长度。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.3 OSD更新功能模块

I.3.1 概述

本模块定义OSD文本内容更新、获取等接口，见表I.3。

表 I.3 OSD 更新功能模块接口

接口	说明
getOsdXmlFile	获取 osd xml 文件下载路径。
getOSDTAGLength	获取相应 tag 对应的文本长度。
getOSDTAGValue	获取相应 tag 对应的文本内容。

I.3.2 常量定义

无。

I.3.3 事件类型

无。

I.3.4 数据结构

无。

I.3.5 回调函数定义

无。

I.3.6 接口定义

I.3.6.1 getOsdXmlFile接口

原型：HI_S32 getOsdXmlFile(HI_CHAR* xml_path, HI_S32 max_length);
功能：获取osd xml文件下载路径。
参数：xml_path--输出参数，获取到的osd xml文件的路径；
max_length--输入参数，xml_path指针指向区域的buff大小。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.3.6.2 getOSDTAGLength接口

原型：HI_S32 getOSDTAGLength(HI_CHAR * tag, HI_S32 *length);
功能：获取相应 tag 对应的文本长度。
参数：tag--输入参数，需要获取的 tag 标识。如：T01；
length--输出参数，返回的文本长度。
返回：HI_S32，操作是否成功，0 表示成功，-1 表示失败。
注：函数主要作用是为 getOSDTAGValue 函数获取 tag 文本分配适当长度的 buff。

I.3.6.3 getOSDTAGValue接口

原型：HI_S32 getOSDTAGValue(HI_CHAR * tag, HI_CHAR *tagValue, HI_S32 length);
功能：获取相应 tag 对应的文本内容。
参数：tag--输入参数，需要获取的 tag 标识。如：T01。
length--输入参数，可获取的文本长度最大值；
tagValue--输出参数，获取到的文本内容。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.4 广告更新功能模块

I.4.1 概述

本模块定义了开机广告接收、广告信息获取、实时广告接收等接口，见表I.4。

表 I.4 广告更新功能模块接口

接口	说明
getAdFinishStatus	开机时获取开机广告处理状态。
getAdInfoCnt	根据广告位获取广告数量。
getAdInfo	根据广告位获取广告内容。
startAdPCT	开始接收实时广告。

I.4.2 常量定义

无。

I.4.3 事件类型

无。

I.4.4 数据结构

I.4.4.1 广告类型标识

原型：typedef enum {

```
    EPGINFO_TYPE_EPG_CONFIG_DAT = 0xf0f0,
    EPGINFO_TYPE_BOOTLOGO = 1,
    EPGINFO_TYPE_MAINMENU,
    EPGINFO_TYPE_CHANNEL_LIST,
    EPGINFO_TYPE_FAV_CHANNEL_LIST,
    EPGINFO_TYPE_EPG_LIST,
    EPGINFO_TYPE_PFBAR,
    EPGINFO_TYPE_VOLUMEBAR,
    EPGINFO_TYPE_AUDIOLOGO,
    EPGINFO_TYPE_ERROR,
} EpgInfoType_e;
```

描述：广告类型标识定义。

成员：

EPGINFO_TYPE_EPG_CONFIG_DAT：配置信息（保留）；

EPGINFO_TYPE_BOOTLOGO：开机logo位置广告；

EPGINFO_TYPE_MAINMENU：主菜单位置广告；

EPGINFO_TYPE_CHANNEL_LIST：频道浏览位置广告；

EPGINFO_TYPE_FAV_CHANNEL_LIST：喜爱频道位置广告；

EPGINFO_TYPE_EPG_LIST：节目指南位置广告；

EPGINFO_TYPE_PFBAR：节目信息位置广告；

EPGINFO_TYPE_VOLUMEBAR：音量条位置广告；

EPGINFO_TYPE_AUDIOLOGO：广播背景广告；

EPGINFO_TYPE_ERROR：错误类型。

I.4.4.2 广告信息结构

原型：typedef struct AdInfo_s {
    HI_U8 mAdPath[256];
    HI_U32 mStartTime;
    HI_U32 mEndTime;
    HI_U8 mDuration;
    HI_U16 mTableExtId;
} AdInfo_t;
描述：广告信息结构定义。
成员：
mAdPath[256]：广告图片所在路径；
mStartTime：广告显示的开始时间（单位秒）；
mEndTime：广告显示的结束时间（单位秒）；
mDuration：广告显示的持续时间，如果为0则一直显示（单位秒）；
mTableExtId：当有多张图片时，按照此值从小到大显示图片。

I.4.5 回调函数定义

无。

I.4.6 接口定义

I.4.6.1 getAdFinishStatus接口

原型：HI_U8 getAdFinishStatus();
功能：开机时获取开机广告处理状态。
参数：无。
返回：HI_S32，开机广告处理是否完成，1表示完成，0表示正在处理。

I.4.6.2 getAdInfoCnt接口

原型：HI_S32 getAdInfoCnt(EpgInfoType_e EpgInfoType, HI_U32 service_id, HI_U32 *AdInfoCnt);
功能：根据参数EpgInfoType类型获取广告数量。
参数：EpgInfoType--输入参数，表示需要获取哪种类型的广告，见EpgInfoType_e；
service_id--输入参数，表示需要获取哪个节目下的广告（仅实时广告使用）；
AdInfoCnt--输出参数，获取到的该类型广告的数量。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.4.6.3 getAdInfo接口

原型：HI_S32 getAdInfo(EpgInfoType_e EpgInfoType, HI_U32 service_id,
AdInfo_t AdInfo[], HI_U32 AdInfoCnt);
功能：根据参数EpgInfoType类型获取广告信息。
参数：EpgInfoType--输入参数，表示需要获取哪种类型的广告，见EpgInfoType_e；
service_id--输入参数，表示需要获取哪个节目下的广告（仅实时广告使用）；
AdInfo[]--输出参数，获取到的广告信息。见AdInfo_t；
AdInfoCnt--输入参数，表示最大能获取多少张广告。

返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.4.6.4 startAdPCT接口

原型：HI_S32 startAdPCT(HI_U16 transport_stream_id);
功能：根据transport_stream_id参数去过滤符合的实时图片数据,一般在切台过程中调用，参数为当前节目所在的tsId。
参数：transport_stream_id--输入参数，当前节目的输入流id。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.5 应急广播监测功能模块

I.5.1 概述

本模块定义了应急广播的接口及消息，见表I.5。

表 I.5 应急广播监测功能模块接口

接口	说明
stopEMBDAction	停止本次应急广播上报。

I.5.2 常量定义

无。

I.5.3 事件类型

无。

I.5.4 数据结构

I.5.4.1 应急广播消息

原型：typedef struct DTH_EMBD_Data_s {
    HI_U16 service_id;
    HI_U16 ts_id;
    HI_U16 orig_net_id;
} DTH_EMBD_Data_t;
描述：应急广播消息数据结构定义。
成员：
service_id：需要触发的应急广播的服务ID;
ts_id：需要触发的应急广播的传输流ID;
orig_net_id：需要触发的应急广播的网络ID。

I.5.5 回调函数定义

无。

I.5.6 接口定义

I.5.6.1 stopEMBDAction接口

原型：HI_S32 stopEMBDAction();

功能：为了保证不漏接应急广播，接收到应急广播之后会一直上报消息，此时调用该接口可以停止本次应急广播的上报。

参数：无。

返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.6 信息服务功能模块

I.6.1 概述

本模块定义了信息服务相关的接口，见表I.6。

表 I.6 信息服务功能模块接口

接口	说明
startDataBD	启动信息服务接收。
getDataBDFinishPercent	获取信息服务下载完成的百分比。
stopDataBD	终止信息服务接收。
deleteDataBDFiles	删除下载的信息服务所有文件。

I.6.2 常量定义

无。

I.6.3 事件类型

无。

I.6.4 数据结构

无。

I.6.5 回调函数定义

无。

I.6.6 接口定义

I.6.6.1 startDataBD接口

原型：HI_S32 startDataBD(HI_CHAR* file_path);

功能：启动信息服务接收。

参数：file_path--输入参数，表示信息服务文件需要保存的路径，如果为 NULL，表明选择默认路径保存。

返回：HI_S32，操作是否成功，0 表示成功，-1 表示失败。

I.6.6.2 getDataBDFinishPercent接口

原型：HI_S32 getDataBDFinishPercent(HI_U32 *precent);
功能：获取信息服务下载完成的百分比。
参数：precent--输出参数，表示已经完成的百分比，取值范围0～100。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.6.6.3 stopDataBD接口

原型：HI_S32 stopDataBD();
功能：终止信息服务接收。
参数：无。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

I.6.6.4 deleteDataBDFiles接口

原型：HI_S32 deleteDataBDFiles();
功能：删除下载的信息服务所有文件，退出信息服务时调用。
参数：无。
返回：HI_S32，操作是否成功，0表示成功，-1表示失败。

附 录 J
（规范性附录）
ATV 组件

J.1 概述

本附录定义了 ATV 组件对外提供的接口，包括频道管理、通道管理和电视设置等功能模块对外接口，见表 J.1。

表 J.1 ATV 组件功能模块

模块	说明
频道管理功能模块	定义了 ATV 频道搜索和频道切换等相关接口。
通道管理功能模块	定义了通道切换及状态查询功能接口。
电视设置功能模块	定义了电视图像、声音、系统、工厂等相关设置接口。

J.2 频道管理功能模块

J.2.1 概述

本模块定义了ATV频道搜索和频道切换等相关接口，见表J.2。

表 J.2 频道管理功能模块接口

接口	说明
hsTVsetChannel	切换频道。
hsTVautoScan	自动搜台功能。
hsTVmanualScan	手动搜台功能。
hsTVstopScan	取消搜台功能。
hsTVsetAudioSystem	设置声音制式。
hsTVgetAudioSystem	获取声音制式。
hsTVsetColorSystem	设置彩色制式。
hsTVgetColorSystem	获取彩色制式。
hsTVgetProgramInfo	获取频道信息。

J.2.2 常量定义

J.2.2.1 ATV搜台频点

原型：const minFreq = 43000;
描述：ATV搜台频点的最小值43Mhz。
原型：const maxFreq = 870000;
描述：ATV搜台频点的最大值870Mhz。

J.2.2.2 ATV搜台回调类型

原型：typedef enum {
    TIL_SCAN_OK = 0x00,
    TIL_SCAN_FAILED,
    TIL_SCAN_CANCELED,
    TIL_SCAN_STARTED = 0x0A,
} til_scan_status_t;
描述：ATV搜台回调类型。
成员：
TIL_SCAN_OK：搜台正常；
TIL_SCAN_FAILED：搜台失败；
TIL_SCAN_CANCELED：搜台取消；
TIL_SCAN_STARTED ：搜台开始。

J.2.2.3 广播媒体类型

原型：typedef enum {
    TIL_BCAST_MEDIUM_UNKNOWN = 0x0,
    TIL_BCAST_MEDIUM_DIGITAL_TERRESTRIAL,
    TIL_BCAST_MEDIUM_DIGITAL_CABLE,
    TIL_BCAST_MEDIUM_DIGITAL_SATELLITE,
    TIL_BCAST_MEDIUM_ANALOG_TERRESTRIAL,
    TIL_BCAST_MEDIUM_ANALOG_CABLE,
    TIL_BCAST_MEDIUM_ANALOG_SATELLITE
} til_broadcast_medium_t;
描述：广播媒体类型。
成员：
TIL_BCAST_MEDIUM_UNKNOWN：未知类型；
TIL_BCAST_MEDIUM_DIGITAL_TERRESTRIAL：数字地面；
TIL_BCAST_MEDIUM_DIGITAL_CABLE：数字有线；
TIL_BCAST_MEDIUM_DIGITAL_SATELLITE：数字卫星；
TIL_BCAST_MEDIUM_ANALOG_TERRESTRIAL：模拟地面；
TIL_BCAST_MEDIUM_ANALOG_CABLE：模拟有线；
TIL_BCAST_MEDIUM_ANALOG_SATELLITE：模拟卫星。

J.2.3 事件类型

无。

J.2.4 数据结构

J.2.4.1 频道信息参数

原型：typedef struct til_channel_s {
    int channelID;
    int frequency;
    int channelNumber;

```
    int originalChannelID;
    char privateData;
    til_broadcast_medium_t broadcastMedium;
} til_channel_t;
```

描述：频道信息参数。

成员：

channelID：频道id；

frequency：频道频率；

channelNumber：频道号；

originalChannelID：原始频道id；

privateData：用户私有信息；

broadcastMedium：广播媒体类型。

J.2.5 回调函数定义

无。

J.2.6 接口定义

J.2.6.1 hsTVsetChannel接口

原型：int hsTVsetChannel(int number);

功能：切换频道。

参数：number--输入参数，频道号。

返回：int，0表示成功，-1表示失败。

J.2.6.2 hsTVautoScan接口

原型：int hsTVautoScan(long lStartFreq , long lEndFreq);

功能：自动搜台功能。

参数：lStartFreq--输入参数， 起始频点；

lEndFreq--输入参数，终止频点。

返回：int，0表示成功，-1表示失败。

J.2.6.3 hsTVmanualScan接口

原型：int hsTVmanualScan(long startfreq, long endfreq);

功能：手动搜台功能。

参数：startfreq--输入参数， 起始频点；

endfreq--输入参数，终止频点。

返回：int，0表示成功，-1表示失败。

J.2.6.4 hsTVstopScan接口

原型：int hsTVstopScan(void);

功能：取消搜台功能。

参数：无。

返回：int，0 表示成功，-1 表示失败。

J.2.6.5 hsTVsetAudioSystem接口

原型：int hsTVsetAudioSystem(int audiosystem);
功能：设置声音制式。
参数：audiosystem--输入参数,声音制式类型。
返回：int，0 表示成功，-1 表示失败。

J.2.6.6 hsTVgetAudioSystem接口

原型：int hsTVgetAudioSystem(int *audiosystem);
功能：获取声音制式。
参数：audiosystem--输出参数,声音制式类型。
返回：int，0 表示成功，-1 表示失败。

J.2.6.7 hsTVsetColorSystem接口

原型：int hsTVsetColorSystem(int colorsystem);
功能：设置彩色制式。
参数：colorsystem--输入参数,彩色制式类型。
返回：int，0 表示成功，-1 表示失败。

J.2.6.8 hsTVgetColorSystem接口

原型：int hsTVgetColorSystem(int *colorsystem);
功能：获取彩色制式。
参数：colorsystem--输出参数,彩色制式类型。
返回：int，0 表示成功，-1 表示失败。

J.2.6.9 hsTVgetProgramInfo接口

原型：int hsTVgetProgramInfo(til_channel_t *channel);
功能：获取频道信息。
参数：channel--输出参数,频道信息。
返回：int，0表示成功，-1表示失败。

J.3 通道管理功能模块

J.3.1 概述

本模块定义了通道切换及状态查询功能接口，见表J.3。

表 J.3 通道管理功能模块接口

接口	说明
hsTVsetInputSource	切换信号源。
hsTVgetCurInputSource	获取当前信号源。
hsTVgetCurInputSourceStatus	获取当前通道的信号状态。

J.3.2 常量定义

J.3.2.1 信号状态类型

原型：typedef enum {
    TIL_INPUT_SOURCE_STATUS_UNKNOWN = 0x0,
    TIL_INPUT_SOURCE_STATUS_CONNECTED,
    TIL_INPUT_SOURCE_STATUS_DISCONNECTED,
    TIL_INPUT_SOURCE_STATUS_VIDEO_UPDATE,
    TIL_INPUT_SOURCE_STATUS_SIGNAL_HDMI,
    TIL_INPUT_SOURCE_STATUS_SIGNAL_DVI,
    TIL_INPUT_SOURCE_STATUS_NO_SIGNAL
} til_input_source_status_t;

描述：信号状态类型。

成员：

TIL_INPUT_SOURCE_STATUS_UNKNOWN：未知信号状态；

TIL_INPUT_SOURCE_STATUS_CONNECTED：信号稳定，有连接；

TIL_INPUT_SOURCE_STATUS_DISCONNECTED：信号未连接；

TIL_INPUT_SOURCE_STATUS_VIDEO_UPDATE：视频信息更新状态；

TIL_INPUT_SOURCE_STATUS_SIGNAL_HDMI：当前信号格式为HDMI；

TIL_INPUT_SOURCE_STATUS_SIGNAL_DVI：当前信号格式为DVI；

TIL_INPUT_SOURCE_STATUS_NO_SIGNAL：信号源有物理连接，但无信号输出。

J.3.2.2 错误类型

原型：typedef enum {
    TIL_OK = 0,
    TIL_NOTPERM = -1,
    TIL_NOTFOUND = -2,
    TIL_IO = -5,
    TIL_TOOBIG = -7,
    TIL_TRYAGAIN = -11,
    TIL_OUTOFMEM = -12,
    TIL_BUSY = -16,
    TIL_NODEVICE = -19,
    TIL_INVALID = -22,
    TIL_ERROR_BASE = -10000,
    TIL_FAIL = -10001
} til_error_code_t;

描述：错误类型。

成员：

TIL_OK：成功；

TIL_NOTPERM：操作不允许；

TIL_NOTFOUND：文件或目录不存在；

TIL_IO：I/O错误；
TIL_TOOBIG：参数太长；
TIL_TRYAGAIN：重试；
TIL_OUTOFMEM：内存溢出；
TIL_BUSY：设备或资源忙；
TIL_NODEVICE：设备不存在；
TIL_INVALID：非法参数；
TIL_ERROR_BASE：基础值；
TIL_FAIL：失败。

J.3.3 事件类型

无。

J.3.4 数据结构

无。

J.3.5 回调函数定义

J.3.5.1 til_input_source_status_cb回调

原型：til_error_code_t (*til_input_source_status_cb)(til_input_source_status_t status);
功能：当前通道的信号状态回调函数，通知上层当前信号状态发生变化。
参数：status--输出参数，信号源状态。
返回：til_error_code_t ，见错误类型。

J.3.6 接口定义

J.3.6.1 hsTVsetInputSource接口

原型：int hsTVsetInputSource(int iInputSource);
功能：切换信号源。
参数：iInputSource--输入参数，需要切入的信号源索引值。
返回：int，1表示切通道成功；0表示切通道失败。

J.3.6.2 hsTVgetCurInputSource接口

原型：int hsTVgetCurInputSource(int *piInputsource);
功能：获取当前信号源。
参数：iInputSource--输出参数，当前信号源索引值。
返回：int，1表示切通道成功；0表示切通道失败。

J.3.6.3 hsTVgetCurInputSourceStatus接口

原型：int hsTVgetCurInputSourceStatus(int iInputsource,
til_input_source_status_t *ptStatus);
功能：获取当前通道的信号状态。
参数：iInputsource--输入参数，当前信号源索引值；

ptStatus--输出参数，信号源状态。

返回：int，1表示获取通道状态成功；0表示获取通道状态失败。

J.4 电视设置功能模块

J.4.1 概述

本模块定义了电视图像、声音、系统、工厂等相关设置接口，见表J.4。

表 J.4 电视设置功能模块接口

接口	说明
SetPictureMode	设置图像模式。
GetPictureMode	获取图像模式。
SetBrightness	设置图像亮度。
GetBrightness	获取图像亮度。
SetContrast	设置图像对比度。
GetContrast	获取图像对比度。
SetSaturation	设置图像饱和度。
GetSaturation	获取图像饱和度。
SetSharpness	设置图像清晰度。
GetSharpness	获取图像清晰度。
SetHue	设置图像色调。
GetHue	获取图像色调。
setColorTemp	设置色温。
getColorTemp	获取色温。
hsTVsetBacklightCustom	设置背光自定义值。
hsTVgetBacklightCustom	获取背光自定义值。
hsTVsetMute	设置静音。
hsTVgetMute	获取当前静音状态。
hsTVsetVolume	设置音量。
hsTVgetVolume	获取当前音量值。
hsTVsetBalance	设置平衡。
hsTVgetBalance	获取当前平衡值。
hsTVsetSoundMode	设置声音模式。
hsTVgetSoundMode	获取当前声音模式。
hsTVsetSpdifMode	设置 SPDIF 输出模式。
hsTVgetSpdifMode	获取当前 SPDIF 输出模式。
hsTVsetAudioEQGainData	设置均衡。
hsTVgetAudioEQGainData	获取当前均衡设置。
hsTVsetAudioSyncDelay	设置音频同步延时。
hsTVgetAudioSyncDelay	获取音频同步延时。

表 J.4（续）

接口	说明
hsTVsetAutoVolumeControl	设置自动音量控制开关。
hsTVgetAutoVolumeControl	获取自动音量控制开关。

J.4.2 常量定义

J.4.2.1 图像模式类型

原型：typedef enum {
    HS_MW_PICMODE_STANDARD = 0x0,
    HS_MW_PICMODE_VIVID = 0x01,
    HS_MW_PICMODE_NATURAL = 0x02,
    HS_MW_PICMODE_CINEMA = 0x04,
    HS_MW_PICMODE_GAME = 0x05,
    HS_MW_PICMODE_DYNAMIC = 0x07
} HS_MW_PICMODE_E;
描述：图像模式类型。
成员：
HS_MW_PICMODE_STANDARD：标准；
HS_MW_PICMODE_VIVID：鲜艳；
HS_MW_PICMODE_NATURAL：自然；
HS_MW_PICMODE_CINEMA：电影；
HS_MW_PICMODE_GAME：游戏；
HS_MW_PICMODE_DYNAMIC：动态。

J.4.2.2 色温类型

原型：typedef enum {
    HS_MW_COLORTEMP_NATURE = 0x0,
    HS_MW_COLORTEMP_COOL = 0x01,
    HS_MW_COLORTEMP_WARM = 0x02,
    HS_MW_COLORTEMP_COLD = 0x04,
    HS_MW_COLORTEMP_HOT = 0x05
} HS_MW_COLORTEMP_E;
描述：色温类型。
成员：
HS_MW_COLORTEMP_NATURE：色温：标准；
HS_MW_COLORTEMP_COOL：色温：偏冷；
HS_MW_COLORTEMP_WARM：色温：偏暖；
HS_MW_COLORTEMP_COLD：色温：冷；
HS_MW_COLORTEMP_HOT：色温：暖。

J.4.3 事件类型

无。

J.4.4 数据结构

无。

J.4.5 回调函数定义

无。

J.4.6 接口定义

J.4.6.1 SetPictureMode接口

原型：int SetPictureMode(HS_MW_PICMODE_E ePictureMode);
功能：设置图像模式。
参数：ePictureMode--输入参数，图像模式类型。
返回：int，1表示正常，0表示错误。

J.4.6.2 GetPictureMode接口

原型：int GetPictureMode(HS_MW_PICMODE_E *pePictureMode);
功能：获取图像模式。
参数：pePictureMode--输出参数，获取的图像模式类型。
返回：int，1表示正常，0表示错误。

J.4.6.3 SetBrightness接口

原型：int SetBrightness(unsigned int u32Brightness);
功能：设置图像亮度。
参数：u32Brightness--输入参数，图像亮度值。
返回：int，1表示正常，0表示错误。

J.4.6.4 GetBrightness接口

原型：int GetBrightness(unsigned int *pu32Brightness);
功能：获取图像亮度。
参数：pu32Brightness--输出参数，获取的图像亮度值。
返回：int，1表示正常，0表示错误。

J.4.6.5 SetContrast接口

原型：int SetContrast(unsigned int u32Contrast);
功能：设置图像对比度。
参数：u32Contrast--输入参数，图像对比度值。
返回：int，1表示正常，0表示错误。

J.4.6.6 GetContrast接口

原型：int GetContrast(unsigned int *pu32Contrast);

功能：获取图像对比度。
参数：pu32Contrast--输出参数，获取的图像对比度值。
返回：int，1表示正常，0表示错误。

J.4.6.7 SetSaturation接口

原型：int SetSaturation(unsigned int u32Saturation);
功能：设置图像饱和度。
参数：u32Saturation--输入参数，图像饱和度值。
返回：int，1表示正常，0表示错误。

J.4.6.8 GetSaturation接口

原型：int GetSaturation(unsigned int *pu32Saturation);
功能：获取图像饱和度。
参数：pu32Saturation--输出参数，获取的图像饱和度值。
返回：int，1表示正常，0表示错误。

J.4.6.9 SetSharpness接口

原型：int SetSharpness(unsigned int u32Sharpness);
功能：设置图像清晰度。
参数：u32Sharpness--输入参数，清晰度值。
返回：int，1表示正常，0表示错误。

J.4.6.10 GetSharpness接口

原型：int GetSharpness(unsigned int *pu32Sharpness);
功能：获取图像清晰度。
参数：pu32Sharpness--输出参数，获取的图像清晰度值。
返回：int，1表示正常，0表示错误。

J.4.6.11 SetHue接口

原型：int SetHue(unsigned int u32Hue);
功能：设置图像色调。
参数：u32Hue--输入参数，色调值。
返回：int，1表示正常，0表示错误。

J.4.6.12 GetHue接口

原型：int GetHue(unsigned int *pu32Hue);
功能：获取图像色调。
参数：pu32Hue--输出参数，获取的图像色调值。
返回：int，1表示正常，0表示错误。

J.4.6.13 setColorTemp接口

原型：int setColorTemp(HS_MW_COLORTEMP_E colorTemp);

功能：设置色温。
参数：colorTemp--输入参数，色温类型。
返回：int，1 表示正常，0 表示错误。

J.4.6.14 getColorTemp接口

原型：int getColorTemp(HS_MW_COLORTEMP_E *colorTemp);
功能：获取色温。
参数：colorTemp--输出参数，获取的色温类型。
返回：int，1 表示正常，0 表示错误。

J.4.6.15 hsTVsetBacklightCustom接口

原型：int hsTVsetBacklightCustom (int custom);
功能：设置背光自定义值。
参数：custom--输入参数，范围：0-30。
返回：int，0 表示成功，-1 表示失败。

J.4.6.16 sTVgetBacklightCustom接口

原型：int hsTVgetBacklightCustom(int *custom);
功能：获取背光自定义值。
参数：custom--输入参数，范围：0-30。
返回：int，0 表示成果，-1 表示失败。

J.4.6.17 hsTVsetMute接口

原型：int hsTVsetMute(bool bMute);
功能：设置静音。
参数：bMut--输入参数，是否静音，1：静音，0：解除静音。
返回：int，0 表示成功，-1 表示失败。

J.4.6.18 hsTVgetMute接口

原型：int hsTVgetMute(bool *pBMute);
功能：获取当前静音状态。
参数：pBMute--返回参数，是否静音，1：静音，0：解除静音。
返回：int，0 表示成功，-1 表示失败。

J.4.6.19 hsTVsetVolume接口

原型：int hsTVsetVolume(int s32Volume);
功能：设置音量。
参数：s32Volume--输入参数，音量值，取值 0～100。
返回：int，0 表示成功，-1 表示失败。

J.4.6.20 hsTVgetVolume接口

原型：hsTVgetVolume(int *pS32Volume);

功能：获取当前音量值。
参数：pS32Volume--返回参数，音量值，取值 0～100。
返回：int，0 表示成功，-1 表示失败。

J.4.6.21 hsTVsetBalance接口

原型：int hsTVsetBalance(int s32Balance);
功能：设置平衡。
参数：s32Balance--输入参数，平衡，取值-10～10。
返回：int，0 表示成功，-1 表示失败。

J.4.6.22 hsTVgetBalance接口

原型：int hsTVgetBalance(int *pS32Balance);
功能：获取当前平衡值。
参数：pS32Balance--返回参数，平衡，取值-10～10。
返回：int，0 表示成功，-1 表示失败。

J.4.6.23 hsTVsetSoundMode接口

原型：int hsTVsetSoundMode(int s32Sndmode);
功能：设置声音模式。
参数：s32Sndmode--输入参数，声音模式。
返回：int，0 表示成功，-1 表示失败。

J.4.6.24 hsTVgetSoundMode接口

原型：int hsTVgetSoundMode(int *pS32Sndmode);
功能：获取当前声音模式。
参数：pS32Sndmode--返回参数，声音模式。
返回：int，0 表示成功，-1 表示失败。

J.4.6.25 hsTVsetSpdifMode接口

原型：int hsTVsetSpdifMode(int s32SpdifMode);
功能：设置 SPDIF 输出模式。
参数：s32SpdifMode--输入参数，SPDIF 模式自动或者 PCM。
返回：int，0 表示成功，-1 表示失败。

J.4.6.26 hsTVgetSpdifMode接口

原型：int hsTVgetSpdifMode(int *pS32SpdifMode);
功能：获取当前 SPDIF 输出模式。
参数：pS32SpdifMode--返回参数，SPDIF 模式。
返回：int，0 表示成功，-1 表示失败。

J.4.6.27 hsTVsetAudioEQGainData接口

原型：int hsTVsetAudioEQGainData(int* pS32EQGainArraySet,int s32EqArrLen);

功能：设置均衡。
参数：pS32EQGainArraySet--输入参数，均衡器参数。
s32EqArrLen--输入参数，均衡器长度。
返回：int，0 表示成功，-1 表示失败。

J.4.6.28 hsTVgetAudioEQGainData接口

原型：int hsTVgetAudioEQGainData(int* pS32EQGainArrayGet,int s32EqArrLen);
功能：获取当前均衡设置。
参数：pS32EQGainArrayGet--返回参数，均衡器设置值；
s32EqArrLen--输入参数，均衡器长度。
返回：int，0 表示成功，-1 表示失败。

J.4.6.29 hsTVsetAudioSyncDelay接口

原型：int hsTVsetAudioSyncDelay(int s32AudioDelay);
功能：设置音频同步延时。
参数：s32AudioDelay--输入参数，音画同步，取值范围-10～10。
返回：int，0 表示成功，-1 表示失败。

J.4.6.30 hsTVgetAudioSyncDelay接口

原型：int hsTVgetAudioSyncDelay(int *pS32AudioDelay);
功能：获取音频同步延时。
参数：pS32AudioDelay--返回参数，音画同步设置。
返回：int，0 表示成功，-1 表示失败。

J.4.6.31 hsTVsetAutoVolumeControl接口

原型：int hsTVsetAutoVolumeControl(bool bOnOff);
功能：设置自动音量控制开关。
参数：bOnOff--输入参数，自动音量控制开关，1 表示打开，0 表示关闭。
返回：int，0 表示成功，-1 表示失败。

J.4.6.32 hsTVgetAutoVolumeControl接口

原型：int hsTVgetAutoVolumeControl(bool *pBOnOff);
功能：获取自动音量控制开关。
参数：pBOnOff--返回参数自动音量控制开关，1 表示打开，0 表示关闭。
返回：int，0 表示成功，-1 表示失败。

附 录 K
（规范性附录）
应用管理组件

K.1 概述

本附录定义了应用管理组件对外提供的接口，包括应用管理功能模块对外接口，见表 K.1。

表 K.1 应用管理组件功能模块

模块	说明
应用管理功能模块	定义了注册、查询和获取应用的功能接口。

K.2 应用管理功能模块

K.2.1 概述

本模块定义了注册、查询和获取应用的功能接口，见表 K.2。

表 K.2 应用管理功能模块接口

接口	说明
registerApp	注册应用进程,告知应用管理服务。
requestApp	请求改变应用状态。
install	安装应用。
uninstall	卸载应用。
removeUserData	删除应用的缓冲。
start	启动应用。
stop	停止应用。
showHome	显示主页屏幕界面。
registerService	注册服务。
setInstallBwList	安装黑白名单。
delInstallBwList	删除黑白名单。
getInstallBwList	获取已经安装的黑白名单。
getPackageInfo	获取安装包信息。
getPackages	获取已安装的应用安装包列表。
getStatus	获取所有应用状态。
getAppStatus	获取指定应用状态。

K.2.2 常量定义

K.2.2.1 启动应用方式

```
原型：enum AppStartFlag {
    APP_START_NONE              = 0x00,
```

```
    APP_START_BACKGROUND        = 0x01,
    APP_START_RESTART           = 0x02,
    APP_START_CONFIRM_START     = 0x04,
    APP_START_QUEUE_ACTIONS     = 0x08,
    APP_START_NO_WINDOW         = 0x10,
    APP_START_NO_FOCUS          = 0x20
};
```

描述：应用启动方式。

成员：

APP_START_NONE：正常启动；

APP_START_BACKGROUND：启动到后台；

APP_START_RESTART：重启；

APP_START_CONFIRM_START：有条件启动，如应用状态从CREATE超过启动时间后，才切换到STARTING；

APP_START_QUEUE_ACTIONS：启动后，通过action反馈返回值；

APP_START_NO_WINDOW：告知appman server，应用是没有窗口，不需要窗口显示；

APP_START_NO_FOCUS：告知appman server，应用是没有焦点。

K.2.2.2 通知应用状态

```
原型：enum AppContextCallback {
   APP_CB_START,
   APP_CB_RESUME,
   APP_CB_PAUSE,
   APP_CB_STOP,
   APP_CB_TERMINATE,
};
```

描述：通知应用状态。

成员：

APP_CB_START：应用处于启动状态；

APP_CB_RESUME：应用处于恢复状态；

APP_CB_PAUSE：应用处于暂停状态；

APP_CB_STOP：应用处于停止状态；

APP_CB_TERMINATE：应用处于中断状态。

K.2.2.3 请求应用行为

```
原型：enum AppNotificationId {
    APP_REQ_STARTED,
    APP_REQ_PAUSE,
    APP_REQ_BACKGROUND,
    APP_REQ_FINISH,
    APP_REQ_STOP,
    APP_REQ_APPSHOW,
    APP_REQ_APPHIDE,
```

```
    APP_REQ_APPFOCUS,
};
```

描述：请求应用行为。

成员：

APP_REQ_STARTED：请求启动；

APP_REQ_PAUSE：请求应用暂停；

APP_REQ_BACKGROUND：请求应用到后台；

APP_REQ_FINISH：完成前台后切换；

APP_REQ_STOP：请求应用停止；

APP_REQ_APPSHOW：请求显示应用；

APP_REQ_APPHIDE：请求隐藏应用；

APP_REQ_APPFOCUS：请求应用成为焦点应用。

K.2.2.4 应用状态信息

```
原型：enum AppState {
    APP_STATE_NONE,
    APP_STATE_CREATED,
    APP_STATE_STARTING,
    APP_STATE_RUNNING,
    APP_STATE_PAUSED,
    APP_STATE_SUSPENDING,
    APP_STATE_BACKGROUND,
    APP_STATE_SUSPENDED,
    APP_STATE_TERMINATING,
    APP_STATE_KILLED,
};
```

描述：应用状态信息。

成员：

APP_STATE_NONE：应用没有启动；

APP_STATE_CREATED：应用已创建，如fork了，但还没有到appman注册；

APP_STATE_STARTING：应用已注册，正准备开始；

APP_STATE_RUNNING：应用处于前台、激活状态；

APP_STATE_PAUSED：应用处于后台，准备要激活；

APP_STATE_SUSPENDING：应用准备进入后台、停止或挂起；

APP_STATE_BACKGROUND：应用处于后台；

APP_STATE_SUSPENDED：应用挂起；

APP_STATE_TERMINATING：应用正中断；

APP_STATE_KILLED：应用已删除，但还没有清除干净。

K.2.3 事件类型

无。

K.2.4 数据结构

K.2.4.1 包信息

```
原型：struct PackageInfo {
    string        pkgId;
    string        pkgName;
    string        pkgPath;
    string        iconPath;
    int           pkgType;
    uid_t         uid;
    gid_t         gid;
    string        pkgVersion;
    int           pkgSize;
} PackageInfo;
```

描述：包信息。

成员：

pkgId：包的唯一id；

pkgName：包名；

pkgPath：安装包位置；

iconPath：安装包图标；

pkgType：包类型；

uid：用户id；

gid：组id；

pkgVersion：包版本；

pkgSize：包大小。

K.2.4.2 应用状态结构

```
原型：struct AppStatus {
    pid_t        pid;
    uid_t        uid;
    gid_t        gid;
    string       appid;
    string       pkgid;
    AppState     state;
};
```

描述：包信息。

成员：

pid：应用进程id；

uid：用户id，用于权限管理；

gid：组id，用于权限管理；

appid：应用id；

pkgid：包id；

state：应用状态。

K.2.5 回调函数定义

K.2.5.1 onStateChange回调

原型：void onStateChange(int cbId);
功能：应用管理服务中，当应用的运行内部状态(start/resume/pause/stop/terminate)改变时，告知应用，便于应用 runtime 处理。
参数：cbId--输入参数，状态类型。
返回：无。

K.2.5.2 onActionObserver回调

原型：int onActionObserver(const std::shared_ptr<Action>& action);
功能：以 Action 方式通知应用回调函数。
参数：action--消息类型，一般用于传递用户私有数据。
返回：int，正常则返回 0。

K.2.6 接口定义

K.2.6.1 registerApp接口

原型：int registerApp(unsigned int flags = 0);
功能：注册应用进程，启动 pool 线程池，处理 binder 事件，告知应用管理服务。
参数：flags--输入参数，应用的启动方式。
返回：int，正常则返回 0；否则返回错误码，注册失败，启动应用失败。

K.2.6.2 requestApp接口

原型：int requestApp(int reqId, int param = 0)
功能：请求改变应用状态。
参数：reqId --输入参数，改变应用状态；
param--输入参数，默认为 0，根据请求状态，带上对应参数。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.3 install接口

原型：int install(const string& path, const string& id, uint32_t flags)
功能：安装应用；返回 0 为成功。
参数：path --输入参数，安装的应用来源的地址；
id --输入参数，安装的应用 id，是用于通知应用的下载与安装状态的标志；
flags --输入参数，，如果 flags 为 0，表示完全新应用；flasg 为 1，表示更新应用。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.4 uninstall接口

原型：int uninstall(const string& pkgid)
功能：卸载应用。
参数：pkgid --输入参数，应用包的 id，是唯一的。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.5 removeUserData接口

原型：int removeUserData(const string& pkgid)
功能：删除应用的缓冲。
参数：pkgid --输入参数，应用包的 id，是唯一的。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.6 start接口

原型：int start(const string& pkgid)
功能：启动应用。
参数：pkgid --输入参数，应用包的 id，是唯一的。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.7 stop接口

原型：int stop(const string& pkgid)
功能：停止应用。
参数：pkgid --输入参数，应用包的 id，是唯一的。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.8 showHome接口

原型：int showHome()
功能：显示主页屏幕界面。
参数：无。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.9 registerService接口

原型：int registerService(const string& name, const sp<IBinder>& service)
功能：注册服务，服务是采用 binder interface。
参数：name --输入参数，服务名称；
service--服务 binder 句柄。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.10 setInstallBwList接口

原型：int setInstallBwList(const string& pkgid, const string& version, bool isBlack)
功能：安装黑白名单。
参数：pkgid --输入参数，应用包的 id，是唯一的；
version --输入参数，应用版本；
isBlack --输入参数，为 true，是黑名单。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.11 delInstallBwList接口

原型：int delInstallBwList(const string& pkgid, const string& version, bool isBlack)

功能：删除黑白名单。
参数：pkgid --输入参数，应用包的 id，是唯一的；
version --输入参数，应用版本；
isBlack --输入参数，为 true，是黑名单。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.12 getInstallBwList接口

原型：int getInstallBwList(const string& pkgid, const string& version, bool *isBlack)
功能：获取已经安装的黑白名单。
参数：pkgid --输入参数，应用包的 id，是唯一的；
version --输入参数，应用版本；
isBlack --输出参数，为 true，是黑名单。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.13 getPackageInfo接口

原型：int getPackageInfo(const string& pkgId, PackageInfo& pkg)
功能：获取安装包信息。
参数：pkgid --输入参数，应用包的 id，是唯一的；
pkg --输出参数，包信息。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.14 getPackages接口

原型：int getPackages(list<PackageInfo> &pkgs);
功能：获取已安装的应用安装包列表。
参数：pkgs --输出参数，安装的包列表。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.15 getStatus接口

原型：int getStatus(list<AppStatus> &apps)
功能：获取所有应用状态。
参数：apps --输出参数，所有应用状态信息。
返回：int，正常则返回 0，否则返回错误码。

K.2.6.16 getAppStatus接口

原型：int getAppStatus(const pid_t pid, AppStatus &status)
功能：获取指定应用状态。
参数：pid --输入参数，应用的进程 pid；
status --输出参数，应用状态信息。
返回：int，正常则返回 0，否则返回错误码。

附 录 L
（规范性附录）
消息管理组件

L.1 概述

本附录定义了消息管理组件对外提供的接口，包括消息管理功能模块对外接口，见表 L.1。

表 L.1 消息管理组件功能模块

模块	说明
消息管理功能模块	定义了消息管理组件抽象层的功能接口。

L.2 消息管理功能模块

L.2.1 概述

本模块定义了消息管理组件抽象层的功能接口，见表L.2。

表 L.2 消息管理功能模块接口

接口	说明
EMAL_AddListener	注册事件的监听器。
EMAL_Notify	发送事件消息。

L.2.2 常量定义

无。

L.2.3 事件类型

无。

L.2.4 数据结构

L.2.4.1 消息类型

原型：typedef enum EventManagerEventId {
    EM_EVENT_ID_DCAS = 0,
    EM_EVENT_ID_DRM = 1,
} EM_EVENT_TYPE_e;
描述：指定消息的类型。
成员：
EM_EVENT_ID_DCAS：DCAS消息事件ID;
EM_EVENT_ID_DRM：DRM消息事件ID。

L.2.5 回调函数定义

L.2.5.1 EMAL_ComponentNotify回调

原型：typedef void (*EMAL_ComponentNotify)(int eventId, char* msg, int msgLength);
功能：消息回调函数。
参数：eventId—消息事件ID，取值范围参考EM_EVENT_TYPE_e；
msg –消息内容；
msgLength –消息长度；
返回：无。

L.2.6 接口定义

L.2.6.1 EMAL_AddListener接口

原型：void EMAL_AddListener(int eventId, EMAL_ComponentNotify cb);
功能：本方法用于注册事件的监听器。
参数：eventId--监听的事件 ID，取值范围参考 EM_EVENT_TYPE_e;
cb--回调函数。
返回：无。

L.2.6.2 EMAL_Notify接口

原型：void EMAL_Notify(int eventId, char* msg, int msgLength);
功能：本方法用于发送事件消息。
参数：eventId--事件 ID，取值范围参考 EM_EVENT_TYPE_e;
msg--消息内容；
msgLength--消息内容的长度。
返回：无。

中 华 人 民 共 和 国
广播电视行业标准
智能电视操作系统
第 5 部分：功能组件接口
GY/T 303.5－2018
*
国家新闻出版广电总局广播电视规划院出版发行
责任编辑：王佳梅
查询网址：www.abp2003.cn
北京复兴门外大街二号
联系电话：（010）86093424　　86092923
邮政编码：100866

版权专有　不得翻印