

OCAP Porting Guide

Final
Version 1.0
July 6, 2007

OCAP Porting Guide
Publication Date: July 6, 2007
Revision: Final

Copyright and reproduction notice ©2004-2007 OCAP Development LLC and Vidiom Systems, Inc., all rights reserved.

Trademarks and licenses  “Java and all Java-based marks” and Sun are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows XP Professional is a trademark of Microsoft Corporation.

This product contains Berkeley Software Distribution (“BSD”) source code which was developed by UC Berkeley and its contributors.

This product contains source code developed by MIT and its contributors.

This product uses the NIST DASE Software Development Environment (NIST DASE software) provided herein is released to any person, company or other legal entity (Experimenter) by the National Institute of Standards and Technology (NIST), an agency of the U.S. Department of Commerce, Gaithersburg MD 20899, USA.

This product uses software from the Apache Software Foundation.

This product uses FreeType2 and is distributed under the FreeType Project License.

This product uses Proguard and Cygnus tools as distributed by the Free Software Foundation, Inc.

This product uses the X license for our implementation of Unicode Compression encoding and decoding routines, currently used for EAS (Cable Emergency Alert) messages.

This product uses an implementation of DirectFB from the Free Software Foundation, Inc.

This product uses NanoXML 2 for Java.

All other product names mentioned herein are trademarks of their respective owners. Please refer to the licenses provided with the software for additional information.

Confidentiality and proprietary notice Confidential and proprietary information subject to non-disclosure agreements with OCAP Development LLC. This information is to be used solely for evaluation of continued and future business agreements with OCAP Development LLC, and shall not be distributed or copied without written permission from OCAP Development LLC.

Warranty disclaimer This publication is provided “As Is” without any warranty of any kind, either express or implied. This includes, but is not limited to, the implied warranty of merchantability, fitness for a particular purpose, or non-infringement.

This publication could include technical inaccuracies or typographical errors. Changes made to this software between documentation releases will be incorporated into new editions of the publication. OCAP Development LLC and/or Vidiom Systems, Inc. may make improvements and/or changes in the products described in this publication at any time.

Contents

1 Overview

| | |
|-----------------------|-----|
| Overview | 1-1 |
| Porting layer | 1-2 |
| JVM | 1-2 |
| Java classes | 1-2 |
| OCAP class libraries | 1-3 |
| OCAP managers | 1-3 |
| JNI Libraries | 1-3 |
| JNI porting layer | 1-3 |
| MPE APIs | 1-3 |
| MPE service managers | 1-3 |
| MPEOS porting layer | 1-3 |
| Host device | 1-3 |
| Hardware requirements | 1-4 |
| Development system | 1-4 |
| Host device | 1-4 |
| Television (optional) | 1-4 |
| Serial cable | 1-4 |
| Ethernet cable | 1-4 |
| Software requirements | 1-5 |
| Development system | 1-5 |
| Host device | 1-5 |
| Contents of packages | 1-6 |
| Porting kit | 1-6 |
| Full source | 1-6 |

2 Installing the Software

| | |
|--------------------------------------|------|
| Overview | 2-1 |
| Installing the porting kit | 2-2 |
| Installing the full source | 2-2 |
| Configuring the development system | 2-4 |
| Installing Java | 2-5 |
| Java2 SDK v1.4.2 | 2-5 |
| Java2 SDK v1.3.1_07 | 2-5 |
| Installing ActivePerl | 2-7 |
| Installing the SDK | 2-8 |
| Installing additional resident fonts | 2-9 |
| Configuring the SDK | 2-11 |

3 Setting Up Your Environment

| | |
|---------------------------------|------|
| Overview | 3-1 |
| Startup configuration sequence | 3-2 |
| Configuring the mpeenv.ini file | 3-3 |
| Port specific | 3-3 |
| Debug | 3-4 |
| Display | 3-7 |
| File system | 3-8 |
| Font | 3-10 |
| Front panel | 3-11 |
| Graphic | 3-12 |
| Manager | 3-12 |
| Memory | 3-13 |
| Network | 3-17 |
| OCAP extensions | 3-18 |
| Security | 3-19 |
| System commands | 3-20 |
| System host | 3-21 |
| User interface | 3-21 |
| OCAP stack | 3-22 |
| Miscellaneous | 3-22 |
| DVR | 3-23 |
| Emergency Alert System (EAS) | 3-25 |
| Host | 3-26 |
| Manager | 3-27 |
| MPE | 3-28 |
| Object carousel | 3-28 |
| Service information | 3-33 |

| | |
|--|------|
| Signaling | 3-39 |
| Time | 3-40 |
| JVM specific | 3-40 |
| Format | 3-40 |
| Properties and values | 3-41 |
| Signaling files | 3-47 |
| Configuring the ait-xxxx.properties file | 3-48 |
| Version | 3-48 |
| Transport protocols | 3-49 |
| Applications | 3-50 |
| Application identifiers | 3-51 |
| Required fields | 3-51 |
| Optional fields | 3-53 |
| External authorization | 3-55 |
| Example file | 3-56 |
| Configuring the xait.properties file | 3-57 |
| Version | 3-57 |
| Transport protocols | 3-57 |
| Applications | 3-59 |
| Application identifiers | 3-59 |
| Required fields | 3-59 |
| Optional fields | 3-62 |
| Services | 3-64 |
| Required fields | 3-65 |
| Optional fields | 3-65 |
| Example file | 3-66 |
| Configuring the hostapp.properties file | 3-67 |
| Version | 3-67 |
| Transport protocols | 3-68 |
| Applications | 3-69 |
| Application identifiers | 3-70 |
| Required fields | 3-70 |
| Optional fields | 3-73 |
| Services | 3-75 |
| Required fields | 3-76 |
| Optional fields | 3-76 |
| Example file | 3-77 |
| Configuring the Monitor Application | 3-78 |
| Signaling the Monitor Application | 3-78 |
| Using the same AppID | 3-78 |
| Smart network Monitor Application | 3-79 |
| hostapp.properties file | 3-79 |

| | |
|-------------------|------|
| mpeenv.ini file | 3-80 |
| signaling support | 3-80 |

4 Build Environment

| | |
|---------------------------------------|------|
| Overview | 4-1 |
| Understanding the directory structure | 4-2 |
| %OCAPROOT% (parent) directory | 4-2 |
| Top-level directories | 4-2 |
| apps (applications) | 4-3 |
| assets | 4-3 |
| bin (binaries) | 4-3 |
| docs | 4-3 |
| extensions | 4-3 |
| java | 4-4 |
| jni | 4-4 |
| jvm | 4-4 |
| mpe | 4-4 |
| other | 4-4 |
| target | 4-6 |
| tools | 4-6 |
| Understanding the software tools | 4-8 |
| Build control tools | 4-8 |
| Ant | 4-8 |
| omake | 4-9 |
| Build construction tools | 4-9 |
| ActivePerl | 4-9 |
| Cygwin | 4-9 |
| Target-specific SDKs | 4-10 |
| Understanding build files | 4-11 |
| build.xml files | 4-11 |
| buildrules.properties files | 4-12 |
| buildrules.mak file | 4-13 |
| sdkrules.mak files | 4-13 |
| buildenv.bat files | 4-13 |
| combuildenv.bat files | 4-14 |
| deploy.properties files | 4-14 |
| Makefile files | 4-14 |
| Generating the build | 4-15 |
| Syntax | 4-15 |
| Build options | 4-15 |
| Build targets | 4-16 |

| | |
|--|------|
| Installing the OCAP stack on the host device | 4-17 |
| Before installing | 4-17 |
| Connecting the hardware | 4-17 |
| Flashing the host device | 4-17 |
| OCAP extensions | 4-18 |
| Jni3pExample OCAP extension | 4-19 |
| Legacy DVR recording extension | 4-20 |
| Understanding the implementation | 4-20 |
| mpeenv.ini Configuration | 4-22 |
| Creating third-party OCAP extensions | 4-24 |

5 MPEOS Overview

| | |
|--|------|
| Overview | 5-1 |
| Strategy | 5-2 |
| Understanding the implementation | 5-3 |
| Basic operating-system services | 5-3 |
| DVR services | 5-4 |
| File system and communication services | 5-5 |
| Graphics services | 5-5 |
| Media services | 5-5 |
| POD services | 5-6 |
| Miscellaneous | 5-6 |
| Operating system-specific header files | 5-8 |
| Configuration | 5-8 |
| Definitions | 5-8 |
| Type definitions | 5-8 |
| Dynamic link library (DLL) | 5-9 |
| Definitions | 5-9 |
| Type definitions | 5-9 |
| Digital Video Recorder (DVR) | 5-9 |
| Definitions | 5-9 |
| Type definitions | 5-10 |
| Error codes | 5-10 |
| Definitions | 5-10 |
| Type definitions | 5-12 |
| Event | 5-12 |
| Definitions | 5-12 |
| Type definitions | 5-13 |
| File system | 5-13 |
| Definitions | 5-13 |
| Type definitions | 5-14 |

| | |
|--------------------------|------|
| Graphics | 5-14 |
| Definitions | 5-14 |
| Type definitions | 5-14 |
| Media | 5-15 |
| Definitions | 5-15 |
| Type definitions | 5-19 |
| Networking | 5-19 |
| Definitions | 5-19 |
| Type definitions | 5-34 |
| Storage | 5-36 |
| Definitions | 5-36 |
| Type definitions | 5-37 |
| Synchronization | 5-37 |
| Definitions | 5-37 |
| Type definitions | 5-38 |
| Thread | 5-38 |
| Definitions | 5-38 |
| Type definitions | 5-40 |
| Time | 5-40 |
| Definitions | 5-40 |
| Type definitions | 5-41 |
| Types | 5-42 |
| Definitions | 5-42 |
| Type definitions | 5-42 |
| Utilities | 5-43 |
| Definitions | 5-43 |
| Type definitions | 5-43 |
| Host-device applications | 5-44 |
| Overview | 5-44 |
| Provisioning | 5-45 |
| Organization and design | 5-45 |
| Third-party software | 5-47 |
| Launching the JVM | 5-48 |

6 MPEOS Testing

| | |
|-------------------------|-----|
| Overview | 6-1 |
| Modifying the test code | 6-4 |
| All | 6-4 |
| Group | 6-4 |
| Individual | 6-5 |
| Event | 6-5 |
| Front panel | 6-5 |

| | |
|--|------|
| Memory | 6-5 |
| Network | 6-6 |
| Synchronization | 6-6 |
| Time | 6-7 |
| Understanding the testing framework | 6-8 |
| Target device requirements for testing | 6-10 |

7 Closed-Captioning API

| | |
|------------------------------------|--------|
| Overview | CAP-1 |
| Definitions | CAP-2 |
| MPE_CC_COLOR | CAP-2 |
| MPE_CC_EMBEDDED_COLOR | CAP-2 |
| Data types and structures | CAP-3 |
| mpe_CcAnalogServiceMap | CAP-3 |
| mpe_CcAnalogServices | CAP-3 |
| mpe_CcAttribType | CAP-4 |
| mpe_CcAttributes | CAP-5 |
| mpe_CcBorderType | CAP-6 |
| mpe_CcColor | CAP-7 |
| mpe_CcDigitalServiceMap | CAP-7 |
| mpe_CcDigitalServices | CAP-8 |
| mpe_CcError | CAP-8 |
| mpe_CcFontSize | CAP-8 |
| mpe_CcFontStyle | CAP-9 |
| mpe_CcOpacity | CAP-10 |
| mpe_CcState | CAP-11 |
| mpe_CcTextStyle | CAP-11 |
| mpe_CcType | CAP-12 |
| mpe_Error | CAP-12 |
| Supported functions | CAP-13 |
| mpeos_ccGetAnalogServices | CAP-14 |
| mpeos_ccGetAttributes | CAP-15 |
| mpeos_ccGetCapability | CAP-16 |
| mpeos_ccGetClosedCaptioning | CAP-18 |
| mpeos_ccGetDigitalServices | CAP-19 |
| mpeos_ccGetSupportedServiceNumbers | CAP-20 |
| mpeos_ccSetAnalogServices | CAP-21 |
| mpeos_ccSetAttributes | CAP-22 |
| mpeos_ccSetClosedCaptioning | CAP-24 |
| mpeos_ccSetDigitalServices | CAP-25 |

8 Common-Download API

| | |
|---------------------------|-------|
| Overview | CDL-1 |
| Error codes | CDL-2 |
| MPE_EINVAL | CDL-2 |
| MPE_SUCCESS | CDL-2 |
| Data types and structures | CDL-3 |
| mpe_Error | CDL-3 |
| mpe_EventQueue | CDL-3 |
| Events | CDL-4 |
| MPE_CDL_EVENT_SHUTDOWN | CDL-4 |
| Supported functions | CDL-5 |
| mpeos_cdlRegister | CDL-6 |
| mpeos_cdlStartDownload | CDL-7 |
| mpeos_cdlUnregister | CDL-8 |

9 Debug API

| | |
|---------------------------------|--------|
| Overview | DBG-1 |
| Logging | DBG-2 |
| Java-level logging | DBG-2 |
| Native C level logging | DBG-2 |
| Logging commands | DBG-3 |
| Logging examples | DBG-3 |
| Definitions | DBG-4 |
| MPEOS_LOG | DBG-4 |
| Error codes | DBG-5 |
| MPE_EINVAL | DBG-5 |
| MPE_SUCCESS | DBG-5 |
| Data types and structures | DBG-6 |
| mpe_Bool | DBG-6 |
| mpe_DbglStatusFormat | DBG-6 |
| mpe_DbglStatusId | DBG-6 |
| mpe_DbglStatusType | DBG-6 |
| mpe_Error | DBG-6 |
| mpe_EventQueue | DBG-6 |
| mpe_logModule | DBG-7 |
| mpeos_g_logControlTbl | DBG-10 |
| Supported functions | DBG-11 |
| dbg_logViaUDP | DBG-12 |
| mpeos_dbgLogUDP | DBG-13 |
| mpeos_dbgStatusGetTypes | DBG-14 |
| mpeos_dbgStatusRegister | DBG-15 |
| mpeos_dbgStatusRegisterInterest | DBG-16 |

| | |
|--|--------|
| <code>mpeos_dbgStatusUnregister</code> | DBG-17 |
| <code>mpeos_dbgStatusUnregisterInterest</code> | DBG-18 |

10 Display API

| | |
|--------------------------------------|--------|
| Overview | DSP-1 |
| Background and video reference model | DSP-2 |
| Screen layout | DSP-2 |
| Background | DSP-2 |
| Graphics | DSP-2 |
| Video | DSP-2 |
| Coordinate system | DSP-4 |
| Surface | DSP-4 |
| Graphic context | DSP-4 |
| Fonts | DSP-5 |
| Error codes | DSP-6 |
| MPE_EINVAL | DSP-6 |
| MPE_SUCCESS | DSP-6 |
| Data types and structures | DSP-7 |
| mpe_Bool | DSP-7 |
| mpe_Displ1394DeviceInfo | DSP-7 |
| mpe_Displ1394Devices | DSP-8 |
| mpe_DispBGIImage | DSP-8 |
| mpe_DispCoherentConfig | DSP-8 |
| mpe_DispDevice | DSP-8 |
| mpe_DispDeviceConfig | DSP-9 |
| mpe_DispDeviceConfigInfo | DSP-9 |
| mpe_DispDeviceInfo | DSP-10 |
| mpe_DispDeviceType | DSP-11 |
| mpe_DispDfcAction | DSP-11 |
| mpe_DispError | DSP-12 |
| mpe_DispOutputPort | DSP-13 |
| mpe_DispOutputPortInfo | DSP-13 |
| mpe_DispOutputPortOption | DSP-14 |
| mpe_DispOutputPortOptionName | DSP-15 |
| mpe_DispOutputPortOptionValue | DSP-16 |
| mpe_DispOutputPortType | DSP-16 |
| mpe_DispScreen | DSP-17 |
| mpe_DispScreenArea | DSP-17 |
| mpe_DispScreenInfo | DSP-18 |
| mpe_Error | DSP-18 |
| mpe_GfxColor | DSP-18 |
| mpe_GfxDimensions | DSP-18 |

| | |
|---|--------|
| <i>mpe_GfxRectangle</i> | DSP-19 |
| <i>mpe_GfxSurface</i> | DSP-19 |
| Events | DSP-20 |
| <i>MPE_DFC_CHANGED</i> | DSP-20 |
| Supported functions | DSP-21 |
| <i>mpeos_dispBGImageDelete</i> | DSP-23 |
| <i>mpeos_dispBGImageGetSize</i> | DSP-24 |
| <i>mpeos_dispBGImageNew</i> | DSP-25 |
| <i>mpeos_dispCheckDFC</i> | DSP-26 |
| <i>mpeos_dispDisplayBGImage</i> | DSP-28 |
| <i>mpeos_dispEnableOutputPort</i> | DSP-29 |
| <i>mpeos_dispFlushGfxSurface</i> | DSP-30 |
| <i>mpeos_dispGetBGColor</i> | DSP-31 |
| <i>mpeos_dispGetCoherentConfigCount</i> | DSP-32 |
| <i>mpeos_dispGetCoherentConfigs</i> | DSP-33 |
| <i>mpeos_dispGetConfigCount</i> | DSP-34 |
| <i>mpeos_dispGetConfigInfo</i> | DSP-35 |
| <i>mpeos_dispGetConfigs</i> | DSP-36 |
| <i>mpeos_dispGetConfigSet</i> | DSP-37 |
| <i>mpeos_dispGetConfigSetCount</i> | DSP-38 |
| <i>mpeos_dispGetCurrConfig</i> | DSP-39 |
| <i>mpeos_dispGetDeviceCount</i> | DSP-40 |
| <i>mpeos_dispGetDeviceInfo</i> | DSP-41 |
| <i>mpeos_dispGetDevices</i> | DSP-42 |
| <i>mpeos_dispGetDFC</i> | DSP-43 |
| <i>mpeos_dispGetGfxSurface</i> | DSP-45 |
| <i>mpeos_dispGetOutputPortCount</i> | DSP-46 |
| <i>mpeos_dispGetOutputPortInfo</i> | DSP-47 |
| <i>mpeos_dispGetOutputPorts</i> | DSP-48 |
| <i>mpeos_dispGetRFBypassState</i> | DSP-49 |
| <i>mpeos_dispGetRFChannel</i> | DSP-50 |
| <i>mpeos_dispGetScreenCount</i> | DSP-51 |
| <i>mpeos_dispGetScreenInfo</i> | DSP-52 |
| <i>mpeos_dispGetScreens</i> | DSP-53 |
| <i>mpeos_dispGetVideoOutputPortOption</i> | DSP-54 |
| <i>mpeos_dispSetBGColor</i> | DSP-56 |
| <i>mpeos_dispSetCoherentConfig</i> | DSP-57 |
| <i>mpeos_dispSetCurrConfig</i> | DSP-58 |
| <i>mpeos_dispSetDFC</i> | DSP-59 |
| <i>mpeos_dispSetRFBypassState</i> | DSP-61 |
| <i>mpeos_dispSetRFChannel</i> | DSP-62 |
| <i>mpeos_dispSetVideoOutputPortOption</i> | DSP-63 |
| <i>mpeos_dispWouldImpact</i> | DSP-65 |

11 Digital Video Recorder (DVR) API

| | |
|---------------------------------|--------|
| Overview | DVR-1 |
| Definitions | DVR-2 |
| MPE_DVR_MAX_NAME_SIZE | DVR-2 |
| MPE_DVR_MAX_PIDS | DVR-2 |
| MPE_DVR_MEDIA_VOL_MAX_PATH_SIZE | DVR-2 |
| MPE_DVR_PID_UNKNOWN | DVR-2 |
| MPE_DVR_POSITIVE_INFINITY | DVR-2 |
| Data types and structures | DVR-3 |
| mpe_Bool | DVR-3 |
| mpe_DispDevice | DVR-3 |
| mpe_DvrBitRate | DVR-3 |
| mpe_DvrBuffering | DVR-3 |
| mpe_DvrConversion | DVR-4 |
| mpe_DvrError | DVR-4 |
| mpe_DvrInfoParam | DVR-5 |
| mpe_DvrMediaStreamType | DVR-7 |
| mpe_DvrPidInfo | DVR-7 |
| mpe_DvrPidTable | DVR-8 |
| mpe_DvrPlayback | DVR-8 |
| mpe_DvrState | DVR-8 |
| mpe_DvrString_t | DVR-9 |
| mpe_DvrTsb | DVR-9 |
| mpe_Error | DVR-9 |
| mpe_EventQueue | DVR-9 |
| mpe_MediaVolume | DVR-9 |
| mpe_MediaVolumeAllowedList | DVR-9 |
| mpe_MediaVolumeEvent | DVR-10 |
| mpe_MediaVolumeInfoParam | DVR-10 |
| mpe_SiElemStreamType | DVR-11 |
| mpe_Storage | DVR-12 |
| mpe_StorageHandle | DVR-12 |
| mpeos_MediaVolume | DVR-12 |
| Events | DVR-13 |
| MPE_DVR_EVT_DEVICE_ERROR | DVR-13 |
| MPE_DVR_EVT_SYSTEM_BUSY | DVR-13 |
| MPE_DVR_EVT_SYSTEM_READY | DVR-13 |
| mpe_DvrEvent | DVR-13 |
| Supported functions | DVR-15 |
| mpeos_dvrFreeRecordingList | DVR-18 |
| mpeos_dvrGet | DVR-19 |

| | |
|-----------------------------------|--------|
| mpeos_dvrGetLowPowerResumeTime | DVR-20 |
| mpeos_dvrGetPlayScales | DVR-21 |
| mpeos_dvrGetRecordingList | DVR-22 |
| mpeos_dvrGetSystemStatus | DVR-23 |
| mpeos_dvrGetTrickMode | DVR-24 |
| mpeos_dvrlsDecodable | DVR-25 |
| mpeos_dvrlsDecryptable | DVR-26 |
| mpeos_dvrMediaVolumeAddAlarm | DVR-27 |
| mpeos_dvrMediaVolumeDelete | DVR-28 |
| mpeos_dvrMediaVolumeGetCount | DVR-29 |
| mpeos_dvrMediaVolumeGetInfo | DVR-30 |
| mpeos_dvrMediaVolumeGetList | DVR-31 |
| mpeos_dvrMediaVolumeNew | DVR-32 |
| mpeos_dvrMediaVolumeRegisterQueue | DVR-34 |
| mpeos_dvrMediaVolumeRemoveAlarm | DVR-35 |
| mpeos_dvrMediaVolumeSetInfo | DVR-36 |
| mpeos_dvrPlaybackChangePids | DVR-38 |
| mpeos_dvrPlaybackGetPids | DVR-39 |
| mpeos_dvrPlaybackGetTime | DVR-40 |
| mpeos_dvrPlaybackSetTime | DVR-41 |
| mpeos_dvrPlaybackStop | DVR-42 |
| mpeos_dvrRecordingDelete | DVR-43 |
| mpeos_dvrRecordingGet | DVR-44 |
| mpeos_dvrRecordingPlayStart | DVR-46 |
| mpeos_dvrResumeFromLowPower | DVR-48 |
| mpeos_dvrSetTrickMode | DVR-49 |
| mpeos_dvrTsbBufferingChangePids | DVR-50 |
| mpeos_dvrTsbBufferingStart | DVR-51 |
| mpeos_dvrTsbBufferingStop | DVR-53 |
| mpeos_dvrTsbChangeDuration | DVR-54 |
| mpeos_dvrTsbConvertChangePids | DVR-55 |
| mpeos_dvrTsbConvertStart | DVR-56 |
| mpeos_dvrTsbConvertStop | DVR-58 |
| mpeos_dvrTsbDelete | DVR-59 |
| mpeos_dvrTsbGet | DVR-60 |
| mpeos_dvrTsbNew | DVR-61 |
| mpeos_dvrTsbPlayStart | DVR-62 |

12 Event API

| | |
|-------------|-------|
| Overview | EVT-1 |
| Error codes | EVT-2 |
| MPE_EEVENT | EVT-2 |
| MPE_EINVAL | EVT-2 |
| MPE_EMUTEX | EVT-2 |

| | |
|---------------------------|-------|
| MPE_ENODATA | EVT-2 |
| MPE_ENOMEM | EVT-2 |
| MPEETIMEOUT | EVT-2 |
| MPE_SUCCESS | EVT-2 |
| Data types and structures | EVT-3 |
| mpe_Error | EVT-3 |
| mpe_Event | EVT-3 |
| mpe_EventQueue | EVT-3 |
| Supported functions | EVT-4 |
| mpeos_eventQueueDelete | EVT-5 |
| mpeos_eventQueueNew | EVT-6 |
| mpeos_eventQueueNext | EVT-7 |
| mpeos_eventQueueSend | EVT-8 |
| mpeos_eventQueueWaitNext | EVT-9 |

13 File-System API

| | |
|---|-------|
| Overview | FIL-1 |
| Porting note | FIL-2 |
| Definitions | FIL-3 |
| File status definitions | FIL-3 |
| MPE_FS_STAT_CREATEDATE | FIL-3 |
| MPE_FS_STAT_EXPDATE | FIL-3 |
| MPE_FS_STAT_ISKNOWN | FIL-4 |
| MPE_FS_STAT_MODDATE | FIL-4 |
| MPE_FS_STAT_ORG_ACCESS | FIL-4 |
| MPE_FS_STAT_PERM | FIL-4 |
| MPE_FS_STAT_PRIOR | FIL-4 |
| MPE_FS_STAT_SIZE | FIL-4 |
| MPE_FS_STAT_TYPE | FIL-4 |
| Broadcast file system file-status definitions | FIL-5 |
| MPE_FS_STAT_CONNECTIONAVAIL | FIL-5 |
| MPE_FS_STAT_MOUNTPATH | FIL-5 |
| MPE_FS_STAT_SOURCEID | FIL-5 |
| Open attribute definitions | FIL-6 |
| MPE_FS_OPEN_APPEND | FIL-6 |
| MPE_FS_OPEN_CAN_CREATE | FIL-6 |
| MPE_FS_OPEN_MUST_CREATE | FIL-6 |
| MPE_FS_OPEN_READ | FIL-6 |
| MPE_FS_OPEN_READWRITE | FIL-6 |
| MPE_FS_OPEN_TRUNCATE | FIL-6 |
| MPE_FS_OPEN_WRITE | FIL-7 |

| | |
|---------------------------------------|--------|
| Operating-system specific definitions | FIL-7 |
| MPE_FS_MAX_PATH | FIL-7 |
| Permission definitions | FIL-8 |
| MPE_FS_PERM_GROUP_EXEC | FIL-8 |
| MPE_FS_PERM_GROUP_READ | FIL-8 |
| MPE_FS_PERM_GROUP_WRITE | FIL-8 |
| MPE_FS_PERM_OWNER_EXEC | FIL-8 |
| MPE_FS_PERM_OWNER_READ | FIL-8 |
| MPE_FS_PERM_OWNER_WRITE | FIL-8 |
| MPE_FS_PERM_WORLD_EXEC | FIL-8 |
| MPE_FS_PERM_WORLD_READ | FIL-9 |
| MPE_FS_PERM_WORLD_WRITE | FIL-9 |
| Priority definitions | FIL-10 |
| MPE_FS_PRIOR_HI | FIL-10 |
| MPE_FS_PRIOR_LOW | FIL-10 |
| MPE_FS_PRIOR_MED | FIL-10 |
| Seek definitions | FIL-11 |
| MPE_FS_SEEK_CUR | FIL-11 |
| MPE_FS_SEEK_END | FIL-11 |
| MPE_FS_SEEK_SET | FIL-11 |
| Type-support definitions | FIL-12 |
| MPE_FS_TYPE_DIR | FIL-12 |
| MPE_FS_TYPE_FILE | FIL-12 |
| MPE_FS_TYPE_OTHER | FIL-12 |
| MPE_FS_TYPE_STREAM | FIL-12 |
| MPE_FS_TYPE_STREAMEVENT | FIL-12 |
| MPE_FS_TYPE_UNKNOWN | FIL-12 |
| Error codes | FIL-13 |
| MPE_FS_ERROR_ALREADY_EXISTS | FIL-13 |
| MPE_FS_ERROR_DEVICE_FAILURE | FIL-13 |
| MPE_FS_ERROR_EOF | FIL-13 |
| MPE_FS_ERROR_EVENTS | FIL-13 |
| MPE_FS_ERROR_FAILURE | FIL-13 |
| MPE_FS_ERROR_INVALID_PARAMETER | FIL-13 |
| MPE_FS_ERROR_INVALID_STATE | FIL-14 |
| MPE_FS_ERROR_NOT_FOUND | FIL-14 |
| MPE_FS_ERROR_NO_MOUNT | FIL-14 |
| MPE_FS_ERROR NOTHING_TO_ABORT | FIL-14 |
| MPE_FS_ERROR_OUT_OF_MEM | FIL-14 |
| MPE_FS_ERROR_READ_ONLY | FIL-14 |
| MPE_FS_ERROR_SUCCESS | FIL-14 |
| MPE_FS_ERROR_TIMEOUT | FIL-14 |

| | |
|---------------------------------------|--------|
| MPE_FS_ERROR_UNKNOWN_URL | FIL-15 |
| MPE_FS_ERROR_UNSUPPORT | FIL-15 |
| <i>Data types and structures</i> | FIL-16 |
| <i>General</i> | FIL-16 |
| mpe_Bool | FIL-16 |
| mpe_Dir | FIL-16 |
| mpe_File | FIL-16 |
| mpe_FileChangeHandle | FIL-16 |
| mpe_FileError | FIL-16 |
| mpe_FileInfo | FIL-17 |
| mpe_FileOpenMode | FIL-19 |
| mpe_FileSeekMode | FIL-19 |
| mpe_FileStatMode | FIL-19 |
| mpe_SiServiceHandle | FIL-20 |
| mpe_Time | FIL-20 |
| <i>Broadcast</i> | FIL-21 |
| mpe_DirInfo | FIL-21 |
| mpe_DirStatMode | FIL-21 |
| mpe_DirUrl | FIL-21 |
| mpe_EventQueue | FIL-22 |
| mpe_SiModulationMode | FIL-22 |
| mpe_Stream | FIL-24 |
| mpe_StreamEventInfo | FIL-24 |
| <i>Normal</i> | FIL-25 |
| mpe_DirEntry | FIL-25 |
| mpeos_filesys_ftable_t | FIL-25 |
| <i>General file system functions</i> | FIL-28 |
| mpeos_filesysGetDefaultDir | FIL-29 |
| mpeos_filesysInit | FIL-30 |
| <i>File system-specific functions</i> | FIL-31 |
| mpeos_dirClose | FIL-33 |
| mpeos_dirCreate | FIL-34 |
| mpeos_dirDelete | FIL-35 |
| mpeos_dirGetUStat | FIL-36 |
| mpeos_dirMount | FIL-37 |
| mpeos_dirOpen | FIL-38 |
| mpeos_dirRead | FIL-39 |
| mpeos_dirRename | FIL-40 |
| mpeos_dirSetUStat | FIL-41 |
| mpeos_dirUnmount | FIL-42 |
| mpeos_fileClose | FIL-43 |
| mpeos_fileDelete | FIL-44 |

| | |
|--------------------------------|--------|
| mpeos_fileGetFStat | FIL-45 |
| mpeos_fileGetStat | FIL-47 |
| mpeos_fileOpen | FIL-49 |
| mpeos_fileRead | FIL-51 |
| mpeos_fileRename | FIL-52 |
| mpeos_fileRemoveChangeListener | FIL-53 |
| mpeos_fileSetChangeListener | FIL-54 |
| mpeos_fileSetFStat | FIL-55 |
| mpeos_fileSetStat | FIL-57 |
| mpeos_fileSync | FIL-59 |
| mpeos_fileWrite | FIL-60 |
| mpeos_init | FIL-61 |

14 File Persistent API

| | |
|--|--------|
| Overview | PER-1 |
| Error codes | PER-2 |
| MPE_FS_ERROR_INVALID_PARAMETER | PER-2 |
| MPE_FS_ERROR_OUT_OF_MEM | PER-2 |
| MPE_FS_ERROR_SUCCESS | PER-2 |
| Data types and structures | PER-3 |
| mpe_FileError | PER-3 |
| mpe_Time | PER-3 |
| mpeos_filesys_ftable_t | PER-3 |
| Supported functions | PER-5 |
| mpeos_persistentGetStatFileName | PER-6 |
| mpeos_persistentGetTempStatFileName | PER-7 |
| mpeos_persistentFileInfoDelete | PER-8 |
| mpeos_persistentFileInfoRename | PER-9 |
| mpeos_persistentSetDefaultFileAttributes | PER-10 |

15 Front Panel API

| | |
|--------------------------|-------|
| Overview | FPL-1 |
| Definitions | FPL-2 |
| MPE_FP_BRIGHTNESS_OFF | FPL-2 |
| MPE_FP_BRIGHTNESS_FULL | FPL-2 |
| MPE_FP_COLOR_BLUE | FPL-2 |
| MPE_FP_COLOR_GREEN | FPL-2 |
| MPE_FP_COLOR_ORANGE | FPL-2 |
| MPE_FP_COLOR_RED | FPL-3 |
| MPE_FP_COLOR_UNSUPPORTED | FPL-3 |
| MPE_FP_COLOR_YELLOW | FPL-3 |
| MPE_FP_INDICATOR_MESSAGE | FPL-3 |
| MPE_FP_INDICATOR_POWER | FPL-3 |

| | |
|---------------------------|--------|
| MPE_FP_INDICATOR_RECORD | FPL-3 |
| MPE_FP_INDICATOR_REMOTE | FPL-3 |
| MPE_FP_INDICATOR_RFBYPASS | FPL-3 |
| MPE_FP_INDICATOR_TEXT | FPL-4 |
| Error codes | FPL-5 |
| MPE_EINVAL | FPL-5 |
| MPE_SUCCESS | FPL-5 |
| Data types and structures | FPL-6 |
| mpe_FpBlinkSpec | FPL-6 |
| mpe_FpCapabilities | FPL-6 |
| mpe_FpScrollSpec | FPL-7 |
| mpe_FpTextPanelMode | FPL-8 |
| Supported functions | FPL-9 |
| mpeos_fpGetCapabilities | FPL-10 |
| mpeos_fpGetIndicator | FPL-11 |
| mpeos_fpGetText | FPL-13 |
| mpeos_fplInit | FPL-15 |
| mpeos_fpSetIndicator | FPL-16 |
| mpeos_fpSetText | FPL-18 |

16 Graphics Manager API

| | |
|-----------------------------|-------|
| Overview | GFX-1 |
| Graphic directory structure | GFX-3 |
| Graphics reference model | GFX-4 |
| Screen layout | GFX-4 |
| Background | GFX-4 |
| Graphics | GFX-4 |
| Video | GFX-4 |
| Coordinate system | GFX-6 |
| Surface | GFX-6 |
| Graphic context | GFX-6 |
| Fonts | GFX-7 |
| General definitions | GFX-8 |
| GFX_LOCK | GFX-8 |
| GFX_UNLOCK | GFX-8 |
| MPE_GFX_UNKNOWN | GFX-8 |
| MPE_GFX_WAIT_INFINITE | GFX-8 |
| mpe_gfxArgbToColor | GFX-8 |
| mpe_gfxGetAlpha | GFX-8 |
| mpe_gfxGetBlue | GFX-9 |
| mpe_gfxGetGreen | GFX-9 |
| mpe_gfxGetRed | GFX-9 |

| | |
|---|--------|
| <i>mpe_gfxRgbToColor</i> | GFX-9 |
| <i>OCAP_KEY_PRESS</i> | GFX-9 |
| <i>OCAP_KEY_RELEASE</i> | GFX-9 |
| <i>Supported data types and structures</i> | GFX-10 |
| <i>General data types and structures</i> | GFX-10 |
| <i>mpe_Bool</i> | GFX-10 |
| <i>mpe_Error</i> | GFX-10 |
| <i>mpe_GfxAlphaChannel</i> | GFX-11 |
| <i>mpe_GfxBitDepth</i> | GFX-12 |
| <i>mpe_GfxColor</i> | GFX-12 |
| <i>mpe_GfxColorFormat</i> | GFX-13 |
| <i>mpe_GfxContext</i> | GFX-14 |
| <i>mpe_GfxDimensions</i> | GFX-14 |
| <i>mpe_GfxError</i> | GFX-14 |
| <i>mpe_GfxEvent</i> | GFX-15 |
| <i>mpe_GfxFont</i> | GFX-16 |
| <i>mpe_GfxFontDesc</i> | GFX-17 |
| <i>mpe_GfxFontFactory</i> | GFX-18 |
| <i>mpe_GfxFontFormat</i> | GFX-18 |
| <i>mpe_GfxFontMetrics</i> | GFX-18 |
| <i>mpe_GfxFontStyle</i> | GFX-19 |
| <i>mpe_GfxPaintMode</i> | GFX-19 |
| <i>mpe_GfxPalette</i> | GFX-20 |
| <i>mpe_GfxPoint</i> | GFX-21 |
| <i>mpe_GfxRectangle</i> | GFX-21 |
| <i>mpe_GfxScreen</i> | GFX-21 |
| <i>mpe_GfxSurface</i> | GFX-21 |
| <i>mpe_GfxSurfaceInfo</i> | GFX-21 |
| <i>mpe_GfxWchar</i> | GFX-23 |
| <i>Context data types and structures</i> | GFX-24 |
| <i>mpeos_GfxContext</i> | GFX-24 |
| <i>os_GfxContext</i> | GFX-24 |
| <i>Font data types and structures</i> | GFX-25 |
| <i>mpeos_GfxFont</i> | GFX-25 |
| <i>os_GfxFont</i> | GFX-26 |
| <i>os_Mutex</i> | GFX-26 |
| <i>Font factory data types and structures</i> | GFX-27 |
| <i>mpeos_GfxFontFactory</i> | GFX-27 |
| <i>Screen data types and structures</i> | GFX-28 |
| <i>mpeos_GfxScreen</i> | GFX-28 |
| <i>os_GfxScreen</i> | GFX-29 |

| | |
|-----------------------------------|--------|
| Surface data types and structures | GFX-30 |
| <i>mpeos_GfxSurface</i> | GFX-30 |
| <i>os_GfxSurface</i> | GFX-33 |
| Supported functions | GFX-34 |
| Context functions | GFX-34 |
| <i>mpeos_gfxContextCreate</i> | GFX-36 |
| <i>mpeos_gfxContextDelete</i> | GFX-37 |
| <i>mpeos_gfxContextNew</i> | GFX-38 |
| <i>mpeos_gfxGetClipRect</i> | GFX-39 |
| <i>mpeos_gfxGetColor</i> | GFX-40 |
| <i>mpeos_gfxGetFont</i> | GFX-41 |
| <i>mpeos_gfxGetOrigin</i> | GFX-42 |
| <i>mpeos_gfxGetPaintMode</i> | GFX-43 |
| <i>mpeos_gfxGetSurface</i> | GFX-45 |
| <i>mpeos_gfxSetClipRect</i> | GFX-46 |
| <i>mpeos_gfxSetColor</i> | GFX-47 |
| <i>mpeos_gfxSetFont</i> | GFX-48 |
| <i>mpeos_gfxSetOrigin</i> | GFX-49 |
| <i>mpeos_gfxSetPaintMode</i> | GFX-50 |
| Drawing functions | GFX-52 |
| <i>mpeos_gfxBitBlt</i> | GFX-54 |
| <i>mpeos_gfxClearRect</i> | GFX-55 |
| <i>mpeos_gfxDrawArc</i> | GFX-56 |
| <i>mpeos_gfxDrawEllipse</i> | GFX-57 |
| <i>mpeos_gfxDrawLine</i> | GFX-58 |
| <i>mpeos_gfxDrawPolygon</i> | GFX-59 |
| <i>mpeos_gfxDrawPolyline</i> | GFX-60 |
| <i>mpeos_gfxDrawRect</i> | GFX-61 |
| <i>mpeos_gfxDrawRoundRect</i> | GFX-62 |
| <i>mpeos_gfxDrawString</i> | GFX-64 |
| <i>mpeos_gfxDrawString16</i> | GFX-66 |
| <i>mpeos_gfxFillArc</i> | GFX-68 |
| <i>mpeos_gfxFillEllipse</i> | GFX-70 |
| <i>mpeos_gfxFillPolygon</i> | GFX-71 |
| <i>mpeos_gfxFillRect</i> | GFX-72 |
| <i>mpeos_gfxFillRoundRect</i> | GFX-73 |
| <i>mpeos_gfxStretchBlt</i> | GFX-74 |
| Font functions | GFX-75 |
| <i>mpeos_gfxFontDelete</i> | GFX-76 |
| <i>mpeos_gfxFontGetList</i> | GFX-77 |
| <i>mpeos_gfxFontHasCode</i> | GFX-78 |
| <i>mpeos_gfxFontNew</i> | GFX-79 |
| <i>mpeos_gfxGetFontMetrics</i> | GFX-81 |
| <i>mpeos_gfxGetCharWidth</i> | GFX-83 |

| | |
|------------------------------|---------|
| mpeos_gfxGetStringWidth | GFX-84 |
| mpeos_gfxGetString16Width | GFX-85 |
| Font-factory functions | GFX-86 |
| mpeos_gfxFontFactoryAdd | GFX-87 |
| mpeos_gfxFontFactoryDelete | GFX-88 |
| mpeos_gfxFontFactoryNew | GFX-89 |
| Screen functions | GFX-90 |
| mpeos_gfxCreateDefaultScreen | GFX-91 |
| mpeos_gfxGetScreen | GFX-92 |
| mpeos_gfxScreenResize | GFX-93 |
| Surface functions | GFX-94 |
| mpeos_gfxPaletteDelete | GFX-95 |
| mpeos_gfxPaletteGet | GFX-96 |
| mpeos_gfxPaletteMatch | GFX-97 |
| mpeos_gfxPaletteNew | GFX-98 |
| mpeos_gfxPaletteSet | GFX-99 |
| mpeos_gfxSurface | GFX-100 |
| mpeos_gfxSurfaceCreate | GFX-101 |
| mpeos_gfxSurfaceDelete | GFX-102 |
| mpeos_gfxSurfaceGetInfo | GFX-103 |
| mpeos_gfxSurfaceNew | GFX-105 |
| User-interface functions | GFX-106 |
| mpeos_gfxWaitNextEvent | GFX-107 |

17 Media API

| | |
|----------------------------------|--------|
| Overview | MED-1 |
| Error codes | MED-2 |
| MPE_EINVAL | MED-2 |
| MPE_ENOMEM | MED-2 |
| MPE_SUCCESS | MED-2 |
| mpe_MediaErrorCode | MED-3 |
| Data types and structures | MED-6 |
| mpe_Bool | MED-6 |
| mpe_DispDevice | MED-6 |
| mpe_Error | MED-6 |
| mpe_EventQueue | MED-6 |
| mpe_GfxDimensions | MED-6 |
| mpe_MediaActiveFormatDescription | MED-7 |
| mpe_MediaAspectRatio | MED-8 |
| mpe_MediaDecodeRequestParams | MED-9 |
| mpe_MediaDecodeSession | MED-9 |
| mpe_MediaDripFeedRequestParams | MED-9 |
| mpe_MediaPid | MED-10 |

| | |
|----------------------------------|--------|
| mpe_MediaPositioningCapabilities | MED-10 |
| mpe_MediaRectangle | MED-11 |
| mpe_MediaTuneParams | MED-11 |
| mpe_MediaTuneRequestParams | MED-11 |
| mpe_MediaTuneType | MED-12 |
| mpe_SiElemStreamType | MED-12 |
| mpe_SiModulationMode | MED-14 |
| Events | MED-17 |
| Common | MED-17 |
| MPE_EVENT_SHUTDOWN | MED-17 |
| Decoding | MED-17 |
| MPE_ACTIVE_FORMAT_CHANGED | MED-17 |
| MPE_ASPECT_RATIO_CHANGED | MED-17 |
| MPE_CONTENT_PRESENTING | MED-18 |
| MPE_DFC_CHANGED | MED-18 |
| MPE_FAILURE_CA_DENIED | MED-18 |
| MPE_FAILURE_NO_DATA | MED-18 |
| MPE_FAILURE_UNKNOWN | MED-18 |
| MPE_STREAM_CA_DENIED | MED-18 |
| MPE_STREAM_CA_UNKNOWN | MED-18 |
| MPE_STREAM_DIALOG_PAYMENT | MED-19 |
| MPE_STREAM_DIALOG_RATING | MED-19 |
| MPE_STREAM_DIALOG_TECHNICAL | MED-19 |
| MPE_STREAM_HW_UNAVAILABLE | MED-19 |
| MPE_STREAM_NO_DATA | MED-19 |
| MPE_STREAM_RETURNED | MED-19 |
| Drip feed | MED-19 |
| MPE_STILL_FRAME_DECODED | MED-19 |
| Tuning | MED-20 |
| MPE_TUNE_ABORT | MED-20 |
| MPE_TUNE_COMPLETE | MED-20 |
| MPE_TUNE_FAIL | MED-20 |
| MPE_TUNE_STARTED | MED-20 |
| Supported functions | MED-21 |
| mpeos_mediaCheckBounds | MED-23 |
| mpeos_mediaDecode | MED-24 |
| mpeos_mediaDripFeedRenderFrame | MED-26 |
| mpeos_mediaDripFeedStart | MED-27 |
| mpeos_mediaDripFeedStop | MED-28 |
| mpeos_mediaFreeze | MED-29 |
| mpeos_mediaFrequencyToTuner | MED-30 |
| mpeos_mediaGetAFD | MED-31 |

| | |
|---------------------------------------|--------|
| mpeos_mediaGetAspectRatio | MED-33 |
| mpeos_mediaGetBounds | MED-34 |
| mpeos_mediaGetInputVideoSize | MED-35 |
| mpeos_mediaGetScaling | MED-36 |
| mpeos_mediaGetTunerInfo | MED-39 |
| mpeos_mediaInit | MED-40 |
| mpeos_mediaRegisterQueueForTuneEvents | MED-41 |
| mpeos_mediaResume | MED-42 |
| mpeos_mediaSetBounds | MED-43 |
| mpeos_mediaShutdown | MED-44 |
| mpeos_mediaStop | MED-45 |
| mpeos_mediaSwapDecoders | MED-46 |
| mpeos_mediaTune | MED-47 |
| mpeos_mediaUnregisterQueue | MED-49 |

18 Memory API

| | |
|------------------------------------|--------|
| Overview | MEM-1 |
| Memory allocation failure handling | MEM-2 |
| Definitions | MEM-4 |
| MPE_MEM_NOPURGE | MEM-4 |
| MPE_MEM_PRIOR_HIGH | MEM-4 |
| MPE_MEM_PRIOR_LOW | MEM-4 |
| mpeos_memAllocP | MEM-4 |
| mpeos_memFreeP | MEM-5 |
| mpeos_memReallocP | MEM-5 |
| Error codes | MEM-6 |
| MPE_EINVAL | MEM-6 |
| MPE_ENODATA | MEM-6 |
| MPE_ENOMEM | MEM-6 |
| MPE_SUCCESS | MEM-6 |
| Data types and structures | MEM-7 |
| mpe_Bool | MEM-7 |
| mpe_Error | MEM-7 |
| mpe_MemColor | MEM-7 |
| mpe_MemHandle | MEM-9 |
| mpe_MemStatsInfo | MEM-9 |
| Supported functions | MEM-11 |
| mpeos_memAllocH | MEM-13 |
| mpeos_memAllocPGen | MEM-14 |
| mpeos_memAllocPProf | MEM-15 |
| mpeos_memCompact | MEM-16 |
| mpeos_memFreeH | MEM-17 |
| mpeos_memFreePGen | MEM-18 |

| | |
|------------------------------------|--------|
| mpeos_memFreePProf | MEM-19 |
| mpeos_memGetFreeSize | MEM-20 |
| mpeos_memGetLargestFree | MEM-21 |
| mpeos_memGetStats | MEM-22 |
| mpeos_memInit | MEM-23 |
| mpeos_memLockH | MEM-24 |
| mpeos_memPurge | MEM-25 |
| mpeos_memReallocH | MEM-26 |
| mpeos_memReallocPGen | MEM-27 |
| mpeos_memReallocPProf | MEM-28 |
| mpeos_memRegisterMemFreeCallback | MEM-29 |
| mpeos_memStats | MEM-31 |
| mpeos_memUnregisterMemFreeCallback | MEM-32 |

19 Module Support API

| | |
|---------------------------|-------|
| Overview | MOD-1 |
| Error codes | MOD-2 |
| MPE_EINVAL | MOD-2 |
| MPE_ENODATA | MOD-2 |
| MPE_SUCCESS | MOD-2 |
| Data types and structures | MOD-3 |
| mpe_Dlmod | MOD-3 |
| mpe_DlmodData | MOD-3 |
| mpe_Error | MOD-3 |
| os_Dlmod and os_DlmodData | MOD-3 |
| os_SymbolTbl | MOD-3 |
| symbolDesc | MOD-4 |
| _symbolDesc | MOD-4 |
| Supported functions | MOD-5 |
| mpeos_dlmodClose | MOD-6 |
| mpeos_dlmodGetSymbol | MOD-7 |
| mpeos_dlmodInit | MOD-8 |
| mpeos_dlmodOpen | MOD-9 |

20 Networking API

| | |
|----------------------------------|-------|
| Overview | NET-1 |
| Definitions | NET-2 |
| Protocol-independent definitions | NET-2 |
| MPE_SOCKET_DGRAM | NET-2 |
| MPE_SOCKET_FD_SETSIZE | NET-2 |
| MPE_SOCKET_FIONBIO | NET-2 |
| MPE_SOCKET_FIONREAD | NET-3 |
| MPE_SOCKET_INVALID_SOCKET | NET-3 |

| | |
|--|--------|
| MPE_SOCKET_MAXHOSTNAMELEN | NET-3 |
| MPE_SOCKET_MSG_OOB | NET-3 |
| MPE_SOCKET_MSG_PEEK | NET-3 |
| MPE_SOCKET_SHUT_RD | NET-3 |
| MPE_SOCKET_SHUT_RDWR | NET-3 |
| MPE_SOCKET_SHUT_WR | NET-4 |
| MPE_SOCKET_STREAM | NET-4 |
| IPv4-specific definitions | NET-5 |
| MPE_SOCKET_AF_INET4 | NET-5 |
| MPE_SOCKET_IN4ADDR_ANY | NET-5 |
| MPE_SOCKET_INET4_ADDRSTRLEN | NET-5 |
| MPE_SOCKET_IN4ADDR_LOOPBACK | NET-5 |
| MPE_SOCKET IPPROTO_IPV4 | NET-5 |
| MPE_SOCKET IPPROTO_TCP | NET-5 |
| MPE_SOCKET IPPROTO_UDP | NET-6 |
| IPv6-specific definitions | NET-7 |
| MPE_SOCKET_IN6ADDR_ANY_INIT | NET-7 |
| MPE_SOCKET_IN6ADDR_LOOPBACK_INIT | NET-7 |
| MPE_SOCKET_INET6_ADDRSTRLEN | NET-7 |
| MPE_SOCKET_AF_INET6 | NET-7 |
| MPE_SOCKET IPPROTO_IPV6 | NET-7 |
| Socket-level socket definition options | NET-8 |
| MPE_SOCKET_SO_BROADCAST | NET-8 |
| MPE_SOCKET_SO_DEBUG | NET-8 |
| MPE_SOCKET_SO_DONTROUTE | NET-8 |
| MPE_SOCKET_SO_ERROR | NET-8 |
| MPE_SOCKET_SO_KEEPALIVE | NET-9 |
| MPE_SOCKET_SO_LINGER | NET-9 |
| MPE_SOCKET_SO_OOBINLINE | NET-9 |
| MPE_SOCKET_SO_RCVBUF | NET-9 |
| MPE_SOCKET_SO_RCVLOWAT | NET-9 |
| MPE_SOCKET_SO_RCVTIMEO | NET-10 |
| MPE_SOCKET_SO_REUSEADDR | NET-10 |
| MPE_SOCKET_SO_SNDBUF | NET-10 |
| MPE_SOCKET_SO SNDLOWAT | NET-10 |
| MPE_SOCKET_SO_SNDTIMEO | NET-10 |
| MPE_SOCKET_SO_TYPE | NET-11 |
| MPE_SOCKET_SOL_SOCKET | NET-11 |
| IPv4-level socket definition options | NET-12 |
| MPE_SOCKET_IPV4_ADD_MEMBERSHIP | NET-12 |
| MPE_SOCKET_IPV4_DROP_MEMBERSHIP | NET-12 |
| MPE_SOCKET_IPV4_MULTICAST_IF | NET-13 |

| | |
|--------------------------------------|--------|
| MPE_SOCKET_IPV4_MULTICAST_LOOP | NET-13 |
| MPE_SOCKET_IPV4_MULTICAST_TTL | NET-13 |
| IPv6-level socket definition options | NET-14 |
| MPE_SOCKET IPV6_ADD_MEMBERSHIP | NET-14 |
| MPE_SOCKET IPV6_DROP_MEMBERSHIP | NET-14 |
| MPE_SOCKET IPV6_MULTICAST_HOPS | NET-15 |
| MPE_SOCKET IPV6_MULTICAST_IF | NET-15 |
| MPE_SOCKET IPV6_MULTICAST_LOOP | NET-15 |
| TCP-level socket definition options | NET-16 |
| MPE_SOCKET TCP_NODELAY | NET-16 |
| Error codes | NET-17 |
| MPE_EINVAL | NET-17 |
| MPE_ENODATA | NET-17 |
| MPE_ENOMEM | NET-17 |
| MPE_ETHREADDEATH | NET-17 |
| MPE_SOCKET_EACCES | NET-18 |
| MPE_SOCKET_EADDRINUSE | NET-18 |
| MPE_SOCKET_EADDRNOTAVAIL | NET-18 |
| MPE_SOCKET_EAFNOSUPPORT | NET-18 |
| MPE_SOCKET_EAGAIN | NET-18 |
| MPE_SOCKET_EALREADY | NET-18 |
| MPE_SOCKET_EBADF | NET-18 |
| MPE_SOCKET_ECONNABORTED | NET-18 |
| MPE_SOCKET_ECONNREFUSED | NET-19 |
| MPE_SOCKET_ECONNRESET | NET-19 |
| MPE_SOCKET_EDESTADDRREQ | NET-19 |
| MPE_SOCKET_EDOM | NET-19 |
| MPE_SOCKET_EHOSTNOTFOUND | NET-19 |
| MPE_SOCKET_EHOSTUNREACH | NET-19 |
| MPE_SOCKET_EINTR | NET-19 |
| MPE_SOCKET_EIO | NET-19 |
| MPE_SOCKET_EISCONN | NET-19 |
| MPE_SOCKET_ELOOP | NET-20 |
| MPE_SOCKET_EMFILE | NET-20 |
| MPE_SOCKET_EMSGSIZE | NET-20 |
| MPE_SOCKET_ENAMETOOLONG | NET-20 |
| MPE_SOCKET_ENETDOWN | NET-20 |
| MPE_SOCKET_ENETUNREACH | NET-20 |
| MPE_SOCKET_ENFILE | NET-20 |
| MPE_SOCKET_ENOBUFS | NET-21 |
| MPE_SOCKET_ENOPROTOOPT | NET-21 |
| MPE_SOCKET_ENORECOVERY | NET-21 |

| | |
|----------------------------------|--------|
| MPE_SOCKET_ENOSPC | NET-21 |
| MPE_SOCKET_ENOTCONN | NET-21 |
| MPE_SOCKET_ENOTSOCK | NET-21 |
| MPE_SOCKET_EOPNOTSUPP | NET-21 |
| MPE_SOCKET_EPIPE | NET-21 |
| MPE_SOCKET_EPROTO | NET-22 |
| MPE_SOCKET_EPROTONOSUPPORT | NET-22 |
| MPE_SOCKET_EPROTOTYPE | NET-22 |
| MPE_SOCKET_ETIMEDOUT | NET-22 |
| MPE_SOCKET_ETRYAGAIN | NET-22 |
| MPE_SOCKET_EWOULDBLOCK | NET-22 |
| <i>Data types and structures</i> | NET-23 |
| <i>General data types</i> | NET-23 |
| mpe_Bool | NET-23 |
| mpe_Socket | NET-23 |
| mpe_SocketFDSet | NET-23 |
| mpe_SocketHostEntry | NET-23 |
| mpe_SocketLinger | NET-24 |
| mpe_SocketSaFamily | NET-24 |
| mpe_SocketSockAddr | NET-24 |
| mpe_SocketSockLen | NET-24 |
| mpe_TimeVal | NET-24 |
| <i>IPv4 data types</i> | NET-25 |
| mpe_SocketIPv4Addr | NET-25 |
| mpe_SocketIPv4McastReq | NET-25 |
| mpe_SocketIPv4SockAddr | NET-25 |
| <i>IPv6 data types</i> | NET-26 |
| mpe_SocketIPv6Addr | NET-26 |
| mpe_SocketIPv6McastReq | NET-26 |
| mpe_SocketIPv6SockAddr | NET-26 |
| <i>Supported functions</i> | NET-27 |
| mpeos_socketAccept | NET-29 |
| mpeos_socketAtoN | NET-31 |
| mpeos_socketBind | NET-32 |
| mpeos_socketClose | NET-34 |
| mpeos_socketConnect | NET-35 |
| mpeos_socketCreate | NET-37 |
| mpeos_socketFDClear | NET-39 |
| mpeos_socketFDIsSet | NET-40 |
| mpeos_socketFDSet | NET-41 |
| mpeos_socketFDZero | NET-42 |
| mpeos_socketGetHostByAddr | NET-43 |

| | |
|--------------------------|--------|
| mpeos_socketGetHostName | NET-45 |
| mpeos_socketGetHostName | NET-46 |
| mpeos_socketGetLastError | NET-47 |
| mpeos_socketGetOpt | NET-48 |
| mpeos_socketGetPeerName | NET-51 |
| mpeos_socketGetSockName | NET-53 |
| mpeos_socketHtoNL | NET-55 |
| mpeos_socketHtoNS | NET-56 |
| mpeos_socketInit | NET-57 |
| mpeos_socketIoclt | NET-58 |
| mpeos_socketListen | NET-59 |
| mpeos_socketNtoA | NET-61 |
| mpeos_socketNtoHL | NET-62 |
| mpeos_socketNtoHS | NET-63 |
| mpeos_socketNtoP | NET-64 |
| mpeos_socketPtoN | NET-65 |
| mpeos_socketRecv | NET-67 |
| mpeos_socketRecvFrom | NET-69 |
| mpeos_socketSelect | NET-71 |
| mpeos_socketSend | NET-74 |
| mpeos_socketSendTo | NET-76 |
| mpeos_socketSetOpt | NET-78 |
| mpeos_socketShutdown | NET-81 |
| mpeos_socketTerm | NET-82 |

21 Point-Of-Deployment (POD) API

| | |
|---------------------------|-------|
| Overview | POD-1 |
| Error codes | POD-2 |
| MPE_EEVENT | POD-2 |
| MPE_EINVAL | POD-2 |
| MPE_ENODATA | POD-2 |
| MPE_ENOMEM | POD-2 |
| MPE_SUCCESS | POD-2 |
| Data types and structures | POD-3 |
| mpe_Bool | POD-3 |
| mpe_Error | POD-3 |
| mpe_EventQueue | POD-3 |
| mpe_Mutex | POD-3 |
| mpe_PODAppInfo | POD-3 |
| mpe_PODDatabase | POD-4 |
| mpe_PODFeatures | POD-4 |
| mpe_PODFeatureParams | POD-5 |

| | |
|------------------------------|--------|
| Events | POD-7 |
| MPE_POD_EVENT_GF_UPDATE | POD-7 |
| MPE_POD_EVENT_APPINFO_UPDATE | POD-7 |
| Supported functions | POD-8 |
| mpeos_podInit | POD-9 |
| mpeos_podGetAppInfo | POD-10 |
| mpeos_podGetCCIBits | POD-11 |
| mpeos_podGetFeatures | POD-12 |
| mpeos_podGetFeatureParam | POD-13 |
| mpeos_podMMIClose | POD-15 |
| mpeos_podMMIConnect | POD-16 |
| mpeos_podReceiveAPDU | POD-17 |
| mpeos_podRegister | POD-18 |
| mpeos_podSASClose | POD-19 |
| mpeos_podSASConnect | POD-20 |
| mpeos_podSendAPDU | POD-21 |
| mpeos_podSetFeatureParam | POD-22 |
| mpeos_podUnregister | POD-24 |

22 Section Filtering API

| | |
|-------------------------------------|-------|
| Overview | FLT-1 |
| Definitions | FLT-3 |
| MPE_SF_INVALID_SECTION_HANDLE | FLT-3 |
| MPE_SF_INVALID_UNIQUE_ID | FLT-3 |
| MPE_SF_OPTION_IF_NOT_CANCELLED | FLT-3 |
| MPE_SF_OPTION_RELEASE_WHEN_COMPLETE | FLT-3 |
| Error codes | FLT-4 |
| MPE_EINVAL | FLT-4 |
| MPE_ENOMEM | FLT-4 |
| MPE_SF_ERROR | FLT-4 |
| MPE_SF_ERROR_FILTER_NOT_AVAILABLE | FLT-4 |
| MPE_SF_ERROR_INVALID_SECTION_HANDLE | FLT-4 |
| MPE_SF_ERROR_SECTION_NOT_AVAILABLE | FLT-4 |
| MPE_SF_ERROR_TUNER_NOT_AT_FREQUENCY | FLT-5 |
| MPE_SF_ERROR_TUNER_NOT_TUNED | FLT-5 |
| MPE_SUCCESS | FLT-5 |
| Data types and structures | FLT-6 |
| mpe_Error | FLT-6 |
| mpe_EventQueue | FLT-6 |
| mpe_FilterComponent | FLT-6 |
| mpe_FilterSectionHandle | FLT-6 |
| mpe_FilterSource | FLT-7 |

| | |
|------------------------------------|--------|
| mpe_FilterSource_INB | FLT-8 |
| mpe_FilterSource_OOB | FLT-8 |
| mpe_FilterSourceParams | FLT-9 |
| mpe_FilterSourceType | FLT-9 |
| mpe_FilterSpec | FLT-10 |
| Event overview | FLT-11 |
| Events | FLT-13 |
| MPE_SF_EVENT_FILTER_AVAILABLE | FLT-13 |
| MPE_SF_EVENT_FILTER_CANCELLED | FLT-13 |
| MPE_SF_EVENT_FILTER_PREEMPTED | FLT-13 |
| MPE_SF_EVENT_LAST_SECTION_FOUND | FLT-13 |
| MPE_SF_EVENT_OUT_OF_MEMORY | FLT-13 |
| MPE_SF_EVENT_SECTION_FOUND | FLT-14 |
| MPE_SF_EVENT_SOURCE_CLOSED | FLT-14 |
| MPE_SF_EVENT_UNKNOWN | FLT-14 |
| Supported functions | FLT-15 |
| mpeos_filterCancelFilter | FLT-16 |
| mpeos_filterCloseDSGTunnel | FLT-17 |
| mpeos_filterGetSectionHandle | FLT-18 |
| mpeos_filterGetSectionSize | FLT-20 |
| mpeos_filterInit | FLT-21 |
| mpeos_filterOpenDSGTunnel | FLT-22 |
| mpeos_filterRegisterAvailability | FLT-23 |
| mpeos_filterRelease | FLT-25 |
| mpeos_filterSectionRead | FLT-26 |
| mpeos_filterSectionRelease | FLT-28 |
| mpeos_filterSetFilter | FLT-29 |
| mpeos_filterShutdown | FLT-32 |
| mpeos_filterUnregisterAvailability | FLT-33 |

23 Sound API

| | |
|---------------------------|-------|
| Overview | SND-1 |
| Error codes | SND-2 |
| MPE_EINVAL | SND-2 |
| MPE_ENOMEM | SND-2 |
| MPE_SUCCESS | SND-2 |
| Data types and structures | SND-3 |
| mpe_Bool | SND-3 |
| mpe_EdHandle | SND-3 |
| mpe_SndDevice | SND-3 |
| mpe_SndError | SND-3 |
| mpe_SndEvent | SND-4 |

| | |
|------------------------------------|--------|
| <i>mpe_SndPlayback</i> | SND-4 |
| <i>mpe_SndSound</i> | SND-4 |
| <i>Supported functions</i> | SND-5 |
| <i>mpeos_sndCreateSound</i> | SND-6 |
| <i>mpeos_sndDeleteSound</i> | SND-7 |
| <i>mpeos_sndGetDeviceCount</i> | SND-8 |
| <i>mpeos_sndGetDevices</i> | SND-9 |
| <i>mpeos_sndGetDevicesForSound</i> | SND-10 |
| <i>mpeos_sndGetMaxPlaybacks</i> | SND-11 |
| <i>mpeos_sndGetTime</i> | SND-12 |
| <i>mpeos_sndInit</i> | SND-13 |
| <i>mpeos_sndPlay</i> | SND-14 |
| <i>mpeos_sndSetTime</i> | SND-15 |
| <i>mpeos_sndStop</i> | SND-16 |

24 Storage Manager API

| | |
|--|--------|
| <i>Overview</i> | STG-1 |
| <i>Definitions</i> | STG-2 |
| <i>MPE_STORAGE_MAX_DISPLAY_NAME_SIZE</i> | STG-2 |
| <i>MPE_STORAGE_MAX_NAME_SIZE</i> | STG-2 |
| <i>MPE_STORAGE_MAX_PATH_SIZE</i> | STG-2 |
| <i>Data types and structures</i> | STG-3 |
| <i>mpe_Bool</i> | STG-3 |
| <i>mpe_EventQueue</i> | STG-3 |
| <i>mpe_StorageError</i> | STG-3 |
| <i>mpe_StorageHandle</i> | STG-4 |
| <i>mpe_StorageInfoParam</i> | STG-4 |
| <i>mpe_StorageStatus</i> | STG-6 |
| <i>mpe_StorageSupportedAccessRights</i> | STG-7 |
| <i>mpeos_Storage</i> | STG-8 |
| <i>Events</i> | STG-9 |
| <i>mpe_StorageEvent</i> | STG-9 |
| <i>Supported functions</i> | STG-10 |
| <i>mpeos_storageEject</i> | STG-11 |
| <i>mpeos_storageGetDeviceCount</i> | STG-12 |
| <i>mpeos_storageGetDeviceList</i> | STG-13 |
| <i>mpeos_storageGetInfo</i> | STG-14 |
| <i>mpeos_storagelInit</i> | STG-16 |
| <i>mpeos_storagelInitializeDevice</i> | STG-17 |
| <i>mpeos_storagelsDetachable</i> | STG-19 |
| <i>mpeos_storagelsDvrCapable</i> | STG-20 |
| <i>mpeos_storagelsRemovable</i> | STG-21 |
| <i>mpeos_storageMakeReadyForUse</i> | STG-22 |

| | |
|---|--------|
| <code>mpeos_storageMakeReadyToDetach</code> | STG-24 |
| <code>mpeos_storageRegisterQueue</code> | STG-25 |

25 Synchronization API

| | |
|------------------------------------|--------|
| Overview | SYN-1 |
| Error codes | SYN-2 |
| MPE_EBUSY | SYN-2 |
| MPE_ECOND | SYN-2 |
| MPE_EINVAL | SYN-2 |
| MPE_EMUTEX | SYN-2 |
| MPE_ENOMEM | SYN-2 |
| MPE_SUCCESS | SYN-2 |
| Data types and structures | SYN-3 |
| <code>mpe_Bool</code> | SYN-3 |
| <code>mpe_Cond</code> | SYN-3 |
| <code>mpe_Error</code> | SYN-3 |
| <code>mpe_Mutex</code> | SYN-3 |
| Supported functions | SYN-4 |
| <code>mpeos_condDelete</code> | SYN-5 |
| <code>mpeos_condGet</code> | SYN-6 |
| <code>mpeos_condNew</code> | SYN-7 |
| <code>mpeos_condSet</code> | SYN-8 |
| <code>mpeos_condUnset</code> | SYN-9 |
| <code>mpeos_condWaitFor</code> | SYN-10 |
| <code>mpeos_mutexAcquire</code> | SYN-11 |
| <code>mpeos_mutexAcquireTry</code> | SYN-12 |
| <code>mpeos_mutexDelete</code> | SYN-13 |
| <code>mpeos_mutexNew</code> | SYN-14 |
| <code>mpeos_mutexRelease</code> | SYN-15 |

26 Thread API

| | |
|-----------------------------|-------|
| Overview | THR-1 |
| Definitions | THR-3 |
| MPE_THREAD_PRIOR_DFLT | THR-3 |
| MPE_THREAD_PRIOR_INC | THR-3 |
| MPE_THREAD_PRIOR_MAX | THR-3 |
| MPE_THREAD_PRIOR_MIN | THR-3 |
| MPE_THREAD_PRIOR_SYSTEM | THR-3 |
| MPE_THREAD_PRIOR_SYSTEM_HI | THR-4 |
| MPE_THREAD_PRIOR_SYSTEM_MED | THR-4 |
| MPE_THREAD_STACK_SIZE | THR-4 |
| MPE_THREAD_STAT_CALLB | THR-4 |

| | |
|----------------------------|--------|
| MPE_THREAD_STAT_COND | THR-4 |
| MPE_THREAD_STAT_DEATH | THR-4 |
| MPE_THREAD_STAT_EVENT | THR-4 |
| MPE_THREAD_STAT_MUTEX | THR-4 |
| Error codes | THR-5 |
| MPE_EINVAL | THR-5 |
| MPE_ENOMEM | THR-5 |
| MPE_SUCCESS | THR-5 |
| Data types and structures | THR-6 |
| mpe_Error | THR-6 |
| mpe_ThreadId | THR-6 |
| mpe_ThreadPrivateData | THR-6 |
| mpe_ThreadStat | THR-6 |
| Supported functions | THR-7 |
| mpeos_threadAttach | THR-8 |
| mpeos_threadCreate | THR-9 |
| mpeos_threadDestroy | THR-11 |
| mpeos_threadGetCurrent | THR-12 |
| mpeos_threadGetData | THR-13 |
| mpeos_threadGetPrivateData | THR-14 |
| mpeos_threadGetStatus | THR-15 |
| mpeos_threadSetData | THR-16 |
| mpeos_threadSetPriority | THR-17 |
| mpeos_threadsetStatus | THR-18 |
| mpeos_threadSleep | THR-19 |
| mpeos_threadYield | THR-20 |

27 Time API

| | |
|---------------------------|-------|
| Overview | TME-1 |
| Error codes | TME-2 |
| MPE_EINVAL | TME-2 |
| MPE_SUCCESS | TME-2 |
| Data types and structures | TME-3 |
| mpe_TimeClock | TME-3 |
| mpe_Error | TME-3 |
| mpe_Time | TME-3 |
| mpe_TimeMillis | TME-3 |
| mpe_TimeVal | TME-3 |
| mpe_TimeTm | TME-3 |
| Supported functions | TME-4 |
| mpeos_timeClock | TME-5 |
| mpeos_timeClockTicks | TME-6 |

| | |
|--------------------------------|--------|
| <i>mpeos_timeClockToMillis</i> | TME-7 |
| <i>mpeos_timeClockToTime</i> | TME-8 |
| <i>mpeos_timeGet</i> | TME-9 |
| <i>mpeos_timeGetMillis</i> | TME-10 |
| <i>mpeos_timeMillisToClock</i> | TME-11 |
| <i>mpeos_timeSystemClock</i> | TME-12 |
| <i>mpeos_timeTimeToClock</i> | TME-13 |
| <i>mpeos_timeTmToTime</i> | TME-14 |
| <i>mpeos_timeToDate</i> | TME-15 |

28 Utility API

| | |
|----------------------------------|--------|
| Overview | UTL-1 |
| Definitions | UTL-2 |
| <i>MPE_BS_MPE_LOWLVL</i> | UTL-2 |
| <i>MPE_BS_MPE_MGRS</i> | UTL-2 |
| <i>MPE_BS_NET_1WAY</i> | UTL-2 |
| <i>MPE_BS_NET_2WAY</i> | UTL-2 |
| <i>MPE_BS_NET_MASK</i> | UTL-2 |
| <i>MPE_BS_NET_NOWAY</i> | UTL-2 |
| Error codes | UTL-3 |
| <i>MPE_EINVAL</i> | UTL-3 |
| <i>MPE_ENOMEM</i> | UTL-3 |
| <i>MPE_SUCCESS</i> | UTL-3 |
| Data types and structures | UTL-4 |
| <i>mpe_Bool</i> | UTL-4 |
| <i>mpe_Error</i> | UTL-4 |
| <i>mpe_EventQueue</i> | UTL-4 |
| <i>mpe_JmpBuf</i> | UTL-4 |
| <i>mpe_STBBootMode</i> | UTL-4 |
| Events | UTL-5 |
| <i>mpe_PowerStatus</i> | UTL-5 |
| Supported functions | UTL-6 |
| <i>mpeos_envGet</i> | UTL-7 |
| <i>mpeos_envInit</i> | UTL-8 |
| <i>mpeos_envSet</i> | UTL-9 |
| <i>mpeos_longJmp</i> | UTL-10 |
| <i>mpeos_registerForPowerKey</i> | UTL-11 |
| <i>mpeos_setJmp</i> | UTL-12 |
| <i>mpeos_stbBoot</i> | UTL-13 |
| <i>mpeos_stbBootStatus</i> | UTL-14 |
| <i>mpeos_stbGetAcOutletState</i> | UTL-16 |
| <i>mpeos_stbGetPowerStatus</i> | UTL-17 |
| <i>mpeos_stbGetRootCerts</i> | UTL-18 |

| | |
|--|--------|
| <code>mpeos_stblsHdCapable</code> | UTL-19 |
| <code>mpeos_stbSetAcOutletState</code> | UTL-20 |

29 Vertical Blanking Interval API

| | |
|--------------------------------------|--------|
| Overview | VBI-1 |
| Definitions | VBI-2 |
| MPE_VBI_OPTION_CLEAR_BUFFER | VBI-2 |
| MPE_VBI_OPTION_CLEAR_READ_DATA | VBI-2 |
| MPE_VBI_OPTION_IF_NOT_STOPPED | VBI-2 |
| Error codes | VBI-3 |
| MPE_EINVAL | VBI-3 |
| MPE_ENOMEM | VBI-3 |
| MPE_SUCCESS | VBI-3 |
| MPE_VBI_ERROR | VBI-3 |
| MPE_VBI_ERROR_FILTER_NOT_AVAILABLE | VBI-3 |
| MPE_VBI_ERROR_INVALID_FILTER_SESSION | VBI-4 |
| MPE_VBI_ERROR_SOURCE_CLOSED | VBI-4 |
| MPE_VBI_ERROR_SOURCE_SCRAMBLED | VBI-4 |
| MPE_VBI_ERROR_UNSUPPORTED | VBI-4 |
| Data types and structures | VBI-5 |
| mpe_DvrPlayback | VBI-5 |
| mpe_Error | VBI-5 |
| mpe_EventQueue | VBI-5 |
| mpe_FilterComponent | VBI-5 |
| mpe_FilterSpec | VBI-6 |
| mpe_MediaDecodeSession | VBI-6 |
| mpe_VBIDataFormat | VBI-7 |
| mpe_VBIFields | VBI-8 |
| mpe_VBIFilterSession | VBI-8 |
| mpe_VBIParameter | VBI-9 |
| mpe_VBISessionParameter | VBI-9 |
| mpe_VBISource | VBI-10 |
| mpe_VBISourceParams | VBI-10 |
| mpe_VBISourceType | VBI-11 |
| Events | VBI-12 |
| MPE_VBI_EVENT_BUFFER_FULL | VBI-12 |
| MPE_VBI_EVENT_DATAUNITS_RECEIVED | VBI-12 |
| Data units | VBI-12 |
| MPE_VBI_EVENT_FILTER_AVAILABLE | VBI-13 |
| MPE_VBI_EVENT_FILTER_STOPPED | VBI-13 |
| MPE_VBI_EVENT_FIRST_DATAUNIT | VBI-13 |
| MPE_VBI_EVENT_OUT_OF_MEMORY | VBI-13 |

| | |
|---------------------------------|---------------|
| MPE_VBI_EVENT_SOURCE_CLOSED | VBI-14 |
| MPE_VBI_EVENT_SOURCE_SCRAMBLED | VBI-14 |
| MPE_VBI_EVENT_UNKNOWN | VBI-14 |
| Supported functions | VBI-15 |
| mpeos_vbiFilterGetParam | VBI-16 |
| mpeos_vbiFilterReadData | VBI-17 |
| mpeos_vbiFilterRelease | VBI-19 |
| mpeos_vbiFilterSetParam | VBI-20 |
| mpeos_vbiFilterStart | VBI-21 |
| mpeos_vbiFilterStop | VBI-24 |
| mpeos_vbiGetParam | VBI-25 |
| mpeos_vbilInit | VBI-26 |
| mpeos_vbiRegisterAvailability | VBI-27 |
| mpeos_vbiShutdown | VBI-29 |
| mpeos_vbiUnregisterAvailability | VBI-30 |

30 Porting the Java Virtual Machine

| | |
|---|------------|
| Overview | A-1 |
| Java's Abstract Windowing Toolkit (AWT) | A-3 |
| OCAP event model | A-3 |
| OCAP focus model | A-4 |
| DVB graphics | A-4 |
| Font factory | A-5 |
| Graphics environment | A-5 |
| File input/output (IO) | A-6 |
| Resource reclamation | A-8 |
| Application context | A-9 |
| Security | A-10 |
| Miscellaneous | A-11 |
| Class file verification | A-11 |
| Time zone | A-11 |
| PBP compliance | A-11 |

Glossary

Index

Preface

Overview

The OpenCable Application Platform (OCAP) Porting Guide provides an outline of the software requirements, the environment requirements, and the procedures for porting the OCAP stack to any platform by implementing the Multimedia Platform Environment Operating System (MPEOS) layer.

Hardware overview

The hardware will be configured based on your platform-specific port. You will need to do the following:

- ◆ Identify the OpenCable-compliant host device (for example, set-top box or cable-ready television) to port.
- ◆ Connect and configure the host device to interface with the development system.
- ◆ Optionally, connect and configure a television or monitor.

Software overview

The Vidiom OCAP porting kit includes:

- ◆ Documentation
- ◆ Header files
- ◆ Source files
- ◆ Tools
- ◆ Linker
- ◆ Sample applications

Software requirements for the development platform vary depending on the host device to which the OCAP stack is being ported. The development system must provide:

- ◆ The ability to compile code
- ◆ The ability to download executables to the target hardware

Software to support these capabilities is often provided by the manufacturer of the target host device in the form of a Software Developer Kit (SDK). The SDK itself might require additional software support. For example, in Vidom's port of the OCAP stack to the Scientific-Atlanta, SA-8300HD set-top box, our development platform was supplied with:

- ◆ PowerTV SDK
- ◆ Windows XP
- ◆ ActivePerl
- ◆ Serial terminal program
- ◆ Source control software

This list is not all-inclusive. Individual developers may also have preferences for software that facilitates their work.

Documentation overview

The *OCAP Porting Guide* describes how to install and configure the hardware/software for porting the OCAP stack to any platform. Most of the examples used in this guide are specific to the Scientific-Atlanta, SA-8300HD set-top box running PowerTV. The following are brief descriptions of all the chapters included in the guide:

- ◆ Chapter 1, *Overview*, provides a description of MPEOS and the OCAP stack, and a list of hardware and software requirements.
- ◆ Chapter 2, *Installing the Software*, provides instructions for installing the porting kit, third-party tools, and the target-specific SDK. It also provides instructions for configuring the development system and the target-specific SDK.
- ◆ Chapter 3, *Setting Up Your Environment*, provides instructions for setting up your runtime environment.
- ◆ Chapter 4, *Build Environment*, provides descriptions of the directory structure of the porting kit, the tools used during the build, the purpose of each component file, and the process for building the OCAP stack.
- ◆ Chapter 5, *MPEOS Overview*, provides the strategy for implementing the MPEOS layer, the directory layout descriptions, the order in which to implement, and the functional requirement information.
- ◆ Chapter 6, *MPEOS Testing*, provides the build and test process for testing and test execution of the MPEOS layer.
- ◆ Application Program Interface (API) sections (these are the remaining sections in the guide) provide detailed description of each function included in the MPEOS layer. These sections are as follows:

- ◆ Chapter CAP, *Closed Captioning API*, provides the captions that are hidden in the video signal and that can be displayed with a special decoder.
- ◆ Chapter CDL, *Common Download API*, provides common download through the CableCARD for downloading new code to the set-top box.
- ◆ Chapter DBG, *Debug API*, provides basic status and diagnostic information messages to the user/developer.
- ◆ Chapter DSP, *Display API*, provides basic discovery and configuration of the display including (potential) support for multiple screens, multiple devices (background, graphics, and video), and multiple device configurations.
- ◆ Chapter DVR, *Digital Video Recorder (DVR) API*, provides functions to create and playback digital video recordings. It also provides time-shift functionality that enables record, pause, rewind, and fast-forward of real-time broadcast content.
- ◆ Chapter EVT, *Event API*, provides queue-based event primitives.
- ◆ Chapter FIL, *File-System API*, provides a portable abstraction layer for the underlying file systems that exist in the target system, such as hard-drives, object carousels, ROM file-systems, and others.
- ◆ Chapter PER, *File Persistent API*, provides support to other ported MPEOS functions. OCAP requires file attributes that are not supported by most file systems -- these functions give porters a common way to incorporate these attributes into their file-system implementations.
- ◆ Chapter FP, *Front Panel API*, provides support for an Xlet with MonitorAppPermission to control the front panel indicators and text display of a set-top box.
- ◆ Chapter GFX, *Graphics Manager API*, provides the basis for support of Java's Abstract Windowing Toolkit (AWT) graphics primitives, event dispatch, and fonts, as well as the HAVi background device.
- ◆ Chapter MED, *Media API*, provides functions for tuning using a source identifier, tuning parameters (frequency, program number, and QAM mode), or tuning to an analog channel given its Electronics Industry Association (EIA) channel number.
- ◆ Chapter MEM, *Memory API*, provides the basic support for allocating and deallocating memory, as well as some additional support for basic system memory status information.
- ◆ Chapter MOD, *Module Support API*, provides support for dynamically linked libraries modules, which are required to support the Java Virtual Machine (JVM) and useful in supporting functional separation within the OpenCable Application Platform (OCAP) stack implementation.

- ◆ Chapter NET, *Networking API*, provides a consistent interface to Berkeley Software Distribution (BSD)/Portable Operating System Interface on Unix (POSIX) style sockets regardless of the underlying operating system.
- ◆ Chapter POD, *Point-Of-Deployment (POD) API*, provides a consistent interface to POD functionality regardless of the underlying operating system.
- ◆ Chapter FLT, *Section Filtering API*, provides support to the service information engine and the Java television APIs.
- ◆ Chapter SND, *Sound API*, provides functions to create and playback digital sound recordings.
- ◆ Chapter STG, *Storage Manager API*, provides functions that notify registered listeners when storage devices are added, removed, or change states.
- ◆ Chapter SYN, *Synchronization API*, provides functions for mutexes, semaphores, and conditions.
- ◆ Chapter THR, *Thread API*, provides support for the Java Virtual Machine (JVM) and the OpenCable Application Platform (OCAP) packages.
- ◆ Chapter TME, *Time API*, provides the current time in various formats.
- ◆ Chapter UTL, *Utility API*, provides support for atomic operations, access to environment variables, support for the canonical `setjmp` and `longjmp` operations, and access to status and configuration information for the platform.
- ◆ Chapter VBI, *Vertical Blanking Interval API*, provides a way to extract VBI data, such as closed captioning, from analog scan lines or MPEG digital video.
- ◆ Appendix A, *Porting the Java Virtual Machine*, provides a description of integration points within a JVM implementation that need to be ported.

Intended audience

Customers for the *OCAP Porting Guide* are the cable set-top box Original Equipment Manufacturers (OEM). This manual is intended to be used by experienced software developers. It assumes a familiarity with C or C++ programming, as well as prior porting experience. A familiarity with developing applications for a television viewing audience is also helpful.

Conventions

This manual uses the following icons to alert you to special information:

- ◆ **NOTE:** This is a note. Notes contain important information that is not procedurally dangerous.
- ◆ **CAUTION:** This is a documentation alert. It contains information about unusual behavior of a utility, command, or procedure.

Cross references provide information about where you can find related information within the manuals provided for OCAP. The cross references are located on the right side of the text.



This is a cross reference to related information.

Definitions of words or phrases that are used in the text are also located on the right side of the text.



This is a definition of a word or phrase found in the text that you may not be familiar with.

In addition, the following typographic conventions and symbols are used in this manual:

- ◆ The names of commands, files, and directories, as well as path statements, environment variables, and on-screen computer output, are indicated by using the following font:

AaBbCc123

The following are examples of this typographical convention:

This definition is found in the psm.h file.

Use ls -a to list all files.

- ◆ The percent signs surrounding an environment variable instructs the system to replace that variable with its value. When you are instructed to type %OCAPROOT%, type all characters as they appear to get the path to work properly.

%ALLCAPS_WITH_PERCENT%

- ◆ The following font indicates that you can replace the name of a variable with an actual name or value:

AaBbCc123

The following is an example of this typographical convention:

Use the *mpegid* parameter to identify the specific Moving Picture Experts Group (MPEG) decoder.

- ◆ Optional elements are contained within square brackets ([]). For example:

sometool [optional]

- ◆ Groupings of elements are enclosed in braces ({}). For example:
`{ CENTER | TOP | BOTTOM }`
- ◆ To repeat the previous item in a syntax statement, ellipses (...) are used. For example:
`sometool [option ...]`
- ◆ A vertical bar (|) is used both for the logical OR in syntax statements and to indicate a choice of two items. For example:
`{ CENTER | TOP | BOTTOM }`

1 Overview

Overview

The OpenCable Application Platform (OCAP) porting kit provides the information that you will need to port the OCAP stack to your hardware's operating system. Once the OCAP stack is ported to your operating system, you can create an OCAP 1.0-compliant host device that will run on any OCAP-compliant network.

The OCAP stack sits on top of the Multimedia Platform Environment Operating System (MPEOS) porting layer, which is the layer you will be working with to port the OCAP stack. The remainder of this guide provides requirements and procedures for creating an MPEOS layer for porting the OCAP stack to your platform. Although the porting process is similar for most platforms, requirements and procedures will vary based on each target port.

Before reading this chapter, you should be familiar with:

- ◆ software development for a cable set-top box or cable-ready television
- ◆ programming with C or C++

After reading this chapter, you should have an understanding of the hardware and software requirements

Porting layer

The porting layer architecture is illustrated in Figure 1-1. The diagram does not attempt to portray literal hierarchical relationships of the individual software components. It merely attempts to show the abstract porting layers as they relate to the major software components.

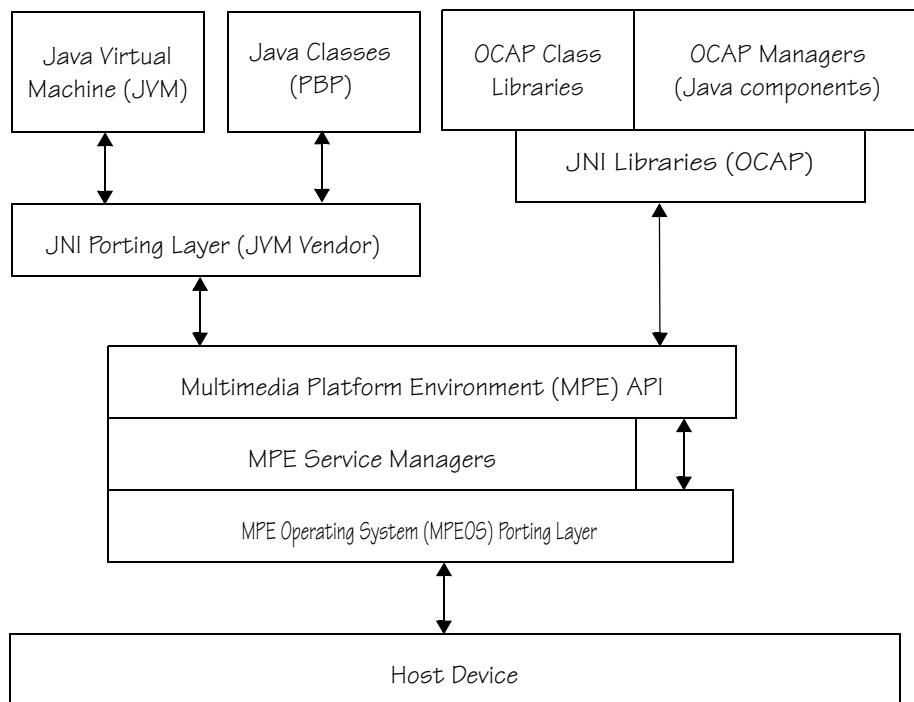


FIGURE 1-1:
Porting layer architecture

JVM

The JVM is the Java Virtual Machine of choice for a particular platform. The OCAP stack implementation is designed to allow for use of any JVM supporting the Java Native Interface (JNI). The JNI interface is used for creation and invocation of the JVM. Every JVM will have slightly different requirements for how it is ported to a new target platform. Preferably, a port of a JVM would use the MPE APIs for additional portability to set-top box platforms.

Java classes

The Java classes consist of the set of Connected Device Configuration (CDC)-Personal Basis Profile (PBP) classes normally provided with the JVM. Most of these classes are completely portable across platforms.

There are generally only two subsets of classes that may require modifications for integration into the OCAP stack environment:

- ◆ The first set includes the class files for `java.awt`. These will likely require mapping to the target platform, which again will ideally use the MPE graphics API.
- ◆ The second set includes some of the support classes for `java.io`, which require modifications for correctly supporting conversion of path references to “.” and the semantics of application file authentication as specified under OCAP.

| | |
|-----------------------------|--|
| OCAP class libraries | The OCAP class libraries are essentially the OCAP API classes defined within the specification. The implementation of these classes is completely portable across all platforms. |
| OCAP managers | The OCAP managers are Java classes that are part of the implementation support for the OCAP APIs. These manager classes entail much of the control logic and state machine support defined or implied within OCAP. The OCAP managers are completely portable across all platforms. |
| JNI Libraries | The JNI libraries provide gateway access between the MPE layer and the OCAP Java managers. The various Java-based components of the OCAP system require bidirectional support from the MPE. The degree of functionality within this layer can be as simple as mapping one API to another or as complex as providing algorithmic support such as data format translation, data encapsulation, and event registry and delivery. |
| JNI porting layer | <p>The JNI porting layer is a thin abstraction allowing the JVM to access common operating system, middleware, and C runtime library operations.</p> <p>If you plan to use the JVM that Vidiom provides with the porting kit, the JNI porting layer may already have been implemented for your target device. If you prefer to use a different JVM, you will likely have to implement the JNI porting layer. To maximize portability to a variety of set-top box platforms, the JNI should be designed to interface with the underlying architecture through the MPE APIs.</p> |
| MPE APIs | The MPE middleware is a collection of platform-independent APIs that provide multimedia-related services. No modification is needed for this layer, but the MPE must be recompiled for the specific target platform. |
| MPE service managers | The MPE service managers provide multimedia-related services. A number of these native MPE managers are very simple and provide only basic support for initialization of the MPE global function dispatch table for the APIs associated with that particular manager. Other managers contain much larger degrees of support for native subsystems. In particular, the file system manager, service information manager, and section filtering manager are examples of native subsystems that contain fairly large and complex implementations of platform independent functionality. |
| MPEOS porting layer | The MPEOS porting layer is where all direct references to the target platform operating system API are isolated. The basic operating system services are abstracted away from the bulk of the MPE code base and are bound at implementation time to the specific target system. Because multiple operating systems must be supported, a set of standard libraries and a typical architecture are used to create a porting layer that minimizes the porting effort to new platforms. The entire layer must be rewritten for each new host device (hardware configuration and operating system combination). |
| Host device | Each port will target a specific hardware platform and operating system. |

Hardware requirements

The hardware requirements vary depending on the development system and the host device being ported. The following requirements are based on the Scientific-Atlanta (SA) SA-8300HD set-top box.

- | | |
|---------------------------|---|
| Development system | <p>The development system must meet the following minimum requirements:</p> <ul style="list-style-type: none"> ◆ Pentium 166 Megahertz (MHz) or faster processor ◆ 64 Megabyte (MB) of physical Random Access Memory (RAM) <hr/> <p>◆ NOTE: Running with less memory may have a severe effect on performance.</p> <ul style="list-style-type: none"> ◆ 900 MB free hard disk space ◆ Compact Disc Read-Only Memory (CD-ROM) drive ◆ 10Base-T Ethernet adapter with an available port ◆ one or two serial ports ◆ Windows XP Professional operating system |
|---------------------------|---|

- | | |
|--------------------|--|
| Host device | <p>The host device must comply with the <i>OpenCable™ Host Device 2.0 Core Functional Requirements</i> specification. The specification defines the range of capabilities and applications to be supported by digital set-top boxes and integrated terminal devices.</p> |
|--------------------|--|



For more information, refer to the OpenCable Web site at <http://www.opencable.com>.

The host device must have the following minimum requirements:

- ◆ 300 MHz processor
- ◆ 28 MB of physical RAM
- ◆ 450 MB free hard disk space
- ◆ 10Base-T Ethernet adapter with an available port
- ◆ one serial port

- | | |
|------------------------------|---|
| Television (optional) | A television or monitor may be required depending on the target platform. |
|------------------------------|---|

- | | |
|---------------------|--|
| Serial cable | A serial cable may be required to connect the development system to the host device. |
|---------------------|--|

- | | |
|-----------------------|---|
| Ethernet cable | An Ethernet cable may be required to connect the host device to the Ethernet network. If the host device does not have an Ethernet port, you can build the operating system and ROM images for serial download. |
|-----------------------|---|

Software requirements

The software requirements vary depending on the target host device and its associated software development kit (SDK).

Development system For the SA 8300HD port, the following software, which is not included in the porting kit, was required on the development system:

- ◆ Windows XP
- ◆ ActivePerl
- ◆ Java Software Development Kit (SDK) v1.4.2
- ◆ Java SDK v1.3.1_07
- ◆ the latest version of the SDK for the target host device

You will need to identify the latest SDK for the target host device. If you do not have the latest version, you will need to get and install the latest version before beginning the port.

Host device The host device must supply an operating system, or equivalent, that can map to the MPEOS functions.

Contents of packages

The OCAP stack is shipped in a full source package and a porting kit package.

Porting kit

The Porting Kit release CD includes the following components:

- ◆ Full source of the MPE and MPEOS layers for the porting layer
- ◆ Binaries of the OCAP Java libraries
- ◆ Full source of the MPE/MPEOS porting layer test suite
- ◆ Full source of the MPE/MPEOS porting layer test framework
- ◆ OCAP Porting Guide
- ◆ Release notes
- ◆ Build tools necessary to compile the MPE/MPEOS layers

◆ **NOTE:** Vidiom does not provide ALL the build tools. For a complete list of tools, refer to the *Understanding the software tools* section.

Full source

The Full Source release CD includes the following components:

- ◆ Full source of the OCAP Java libraries
- ◆ Full source and binaries of the Java integration tests
- ◆ Full source and binaries of the JUnit tests
- ◆ OCAP Porting Kit
- ◆ Release notes
- ◆ Build tools necessary to compile the OCAP Java libraries

◆ **NOTE:** Vidiom does not provide ALL the build tools. For a complete list of tools, refer to the *Understanding the software tools* section.

2 Installing the Software

Overview

Porting the OpenCable Application Platform (OCAP) stack requires a development system and a host device. This section provides the installation and configuration procedures needed on the development system for porting the OCAP stack. It also provides an overview on installing the OCAP stack on the host device.

This chapter covers the installation procedures for the following:

- ◆ the porting kit
- ◆ the SDK

The development system holds all the files required for building the OCAP stack. Development must be done on this computer.

The procedures described in this chapter are based on Vidom's port of the OCAP stack to the Scientific-Atlanta (SA) SA-8300HD set-top box.

Before starting these procedures, you should have:

- ◆ a development system running Windows XP professional
- ◆ an Internet connection available (for downloading supporting software)
- ◆ an available host device

After reading this chapter, you should have:

- ◆ the OCAP Porting Kit installed on your development system
- ◆ the required software installed on your development system

Installing the porting kit

The OCAPI Porting Kit release is distributed as a self-extracting Zip archive. Use the following procedure to install the porting kit onto your development system:

1. Insert the OCAPI installation CD into your computer. If you have Autorun enabled for your CD-ROM, setup will run automatically. If Autorun is not enabled, navigate to the CD-ROM drive.
2. Double-click on the OCAPI-PK.exe file. Once selected, the licensing agreement appears.
3. Use the scroll bar to read the agreement.
4. Click the Accept button to accept the licensing agreement and continue with the installation.

—or—

Click the Decline button to exit the installation.

5. Click the Install button to install the OCAPI stack in the C:\OCAPI-PK directory, which is the default installation directory.
6. To select a different installation directory you will need to:
 - a. Click the Browse button.
 - b. Select the install directory.
 - c. Click the Install button.

Installing the full source

The OCAPI Full Source release is distributed as a self-extracting Zip archive. Use the following procedure to install the full source onto your development system:

1. Insert the OCAPI installation CD into your computer. If you have Autorun enabled for your CD-ROM, setup will run automatically. If Autorun is not enabled, navigate to the CD-ROM drive.
2. Double-click on the OCAPI-FS.exe file. Once selected, the licensing agreement appears.
3. Use the scroll bar to read the agreement.
4. Click the Accept button to accept the licensing agreement.

—or—

Click the Decline button to exit the installation.

5. Click the Install button to install the OCAPI stack in the C:\OCAPI-FS directory, which is the default installation directory.

6. To select a different installation directory you will need to:
 - a. Click the Browse button.
 - b. Select the install directory.
 - c. Click the Install button.

Configuring the development system

Before you begin porting, you must set a few environment variables on the development system:

1. Add an environment variable named OCAPROOT to your environment. As the value for this environment variable, specify the path to your OCAP root directory. For example, on a Windows system where the porting kit is installed to a directory named Vidiom in the Program Files folder, the value of %OCAPROOT% would be C:\Program Files\Vidiom\OCAP1-PK.

◆ **NOTE:** To modify environment variables in Windows, right-click on the My Computer icon and select Properties from the pop-up menu. Select the Advanced tab and click on the Environment Variables button.

2. Add an environment variable named OCAPTC to your environment. This variable points to the directory of the target configuration for the build process. The value of this environment variable should be therelativepathfrom%OCAPROOT%\target\Vidiotomyourdefaulttarget configuration directory. For example, if the target configuration is the debug configuration for a hypothetical target platform XYZ and target host device 123, %OCAPTC% would be set to XYZ\123\debug.
3. Add %OCAPROOT%\tools\Win32\bin to your path statement to provide access to the omake development tool from any command shell.
4. Add or update your environment variable named ComSpec. ComSpec must point to CMD.EXE, which is typically located in the C:\WINDOWS\SYSTEM32 directory, or another shell that properly handles the \C and \K command line options.
5. Set the TMP and TEMP environment variables to a location (for example, C:\TMP and C:\TEMP) where temporary files may be created by tools within the build environment.

Installing Java

There are two Java2 Software Development Kits (SDK) used for compiling the OCAP Java sources and for running Ant on the development system. You must install these SDKs on the development system into specified directories. It is important to specify the correct directory because some build files have hard-coded references to that location.



For details on how to install the Java SDKs, refer to the online installation instructions on the same site as the downloads.

Java2 SDK v1.4.2

To install the Java2 SDK v1.4.2, you need to do the following:

1. Go to the Java Web site at:

<http://java.sun.com/products/archive/j2se/1.4.2/index.html>

2. Click on the SDK Download button. The License Agreement appears.
3. Use the scroll bar to read the agreement.
4. To continue with the installation you will need to:
 - a. Select the Accept button.
 - b. Click the Continue button.
5. Select the platform to download for the development system. For example, when developing for PowerTV use Windows.
6. Save the download in a temporary directory.
7. Double-click on the `j2sdk-1_4_1_02-windows-i586.exe` to install the software. The Destination Location screen appears.
8. Install the Java2 SDK into the `%OCAPROOT%\tools\Win32\j2sdk1.4.2` directory.



For details about the installation, refer to the online Java Installation Instructions.

Java2 SDK v1.3.1_07

To install the Java2 SDK v1.3.1_07, you need to do the following:

1. Go to the Java Web site at:

http://java.sun.com/products/archive/j2se/1.3.1_07/index.html

2. Click on the SDK Download button. The License Agreement appears.
3. Use the scroll bar to read the agreement.
4. To continue with the installation you will need to:
 - a. Select the Accept button.
 - b. Click the Continue button.
5. Select the platform to download for the development system. For example, when developing for PowerTV use Windows.
6. Save the download in a temporary directory.

7. Double-click on the j2sdk-1_3_1_07-windows-i586.exe file to install the software. The Destination Location screen appears.
8. Install the Java2 SDK into the %OCAPROOT%\tools\Win32\jdk1.3.1_07 directory.



For details about the installation, refer to the online Java Installation Instructions.

Installing ActivePerl

ActivePerl v5.8 is required on the development system and is used for running scripts and generating files used by the build environment. ActivePerl is available for Linux, Solaris, and Windows. ActivePerl is provided by ActiveState as a free distribution. To install ActivePerl, you need to do the following:



For more information on ActivePerl, refer to the ActivePerl Web site at www.activestate.com.

1. Go to the ActiveState Web site at:

<http://www.activestate.com>

2. Navigate to the ActivePerl download page.
3. Click on the appropriate version of ActivePerl for your development environment. The File Download window appears.

◆ **NOTE:** The target development environment may be different than the reference platform and may require a different ActivePerl version.

4. Click the Save button.
5. Select a temporary directory in which to save the download.
6. Click the Save button again once the directory is selected. An installer package (for example, ActivePerl-5.8.8.819-MSWin32-x86.msi) is downloaded into the specified directory.
7. Double-click on the installer file to start the setup.
8. Accept all the default setup values. ActivePerl is automatically installed into the c:\Perl directory. Installing ActivePerl outside the OCAP root directory is done to make the un-installation of the two products cleaner.

Installing the SDK

The latest version of the SDK associated with the target host device needs to be installed on a development system in the `%OCAPROOT%\tools\Win32\Host Device OS\SDK name and version` directory. For example, multiple revisions of a hypothetical SDK called `SDK_1.2.3` for a platform ABC might be located in `%OCAPROOT%\tools\Win32\ABC\SDK_1.2.3`, where the `1.2.3` might specify the version number of the SDK.



For more information on installing the latest SDK for the target platform, refer to the host device documentation.

Installing additional resident fonts

The Tiresias font was designed to be viewed on a television screen and is required by OCAP as a resident font. Tiresias is the only font included in the OCAP stack. Many other fonts appear to flicker, faint, or appear missing portions of letters on standard television screens because of the size, height, width, and line thickness. However, high-definition televisions and services correct this issue.



For more information on installing the latest SDK for the target platform, refer to the host device documentation.

To install additional fonts, you need to do the following:

-
- ◆ **NOTE:** This installation is used only for installing additional resident fonts, not for application-installed fonts.
-

1. Copy each font as a .pfr file into the *OCAP-1.0\bin\Vidiom\Host Device Operating System\Host Device\Type of Build\sys\fonts* directory.

- ◆ **NOTE:** Tiresias is located in the *OCAP-1.0\assets\fonts\Bitstream\Tiresias* directory, but it gets copied into the target (bin) directory during the build process.
-

2. Add the new fonts names to the initialization file (*mpeenv.ini*) located in the *OCAP-1.0\bin\Vidiom\HostDeviceOperatingSystem\HostDevice\Type of Build* directory. You can use the font port-specific fields to configure the built-in system fonts. Example font fields are:

`SYSFONT=fontName[,fontName,[...]]`
uniquely defines the system fonts. Where:

fontName
specifies the logical name of the font.

For example:

`SYSFONT=arial,tiresias`

-
- ◆ **NOTE:** Always define the mandatory OCAP font Tiresias as the last font listed in SYSFONT.
-

`SYSFONT.fontName[.style][.n]=fontFile,
[minSize[-maxSize]]`
uniquely defines a specific associated system font and attributes. Where:
fontName
specifies the logical name of the font, as specified in the SYSFONT field.

style specifies the optional style type. Possible values for *style* are:

PLAIN specifies that the font is not bold and not italic. PLAIN is the default.

BOLD specifies a bold font.

ITALIC specifies the font is italicized.

BOLD_ITALIC
specifies a bold and italic font.

n defines an optional identifier. There are five maximum identifiers ranging from 0 to 4. Use the identifier when you have two or more *fontName* or *fontName* plus *style* definitions. If no identifier is defined, the identifier is 0.

fontFile
specifies the absolute path to the font file.

minSize
specifies the optional minimum font size in points for the logical font to support. If the font size is defined, *minSize* is required. If *minSize* is not defined, the font sizes defined in the font file are used.

maxSize
specifies the optional maximum font size in points.

For example:

```
SYSFONT.Tiresias.BOLD.1=/romfs/sys/fonts/  
Tires-o_802.pfr,10-48
```

-
- ◆ **NOTE:** Test the new fonts using a simple application (for example, Hello World) to the screen before writing any complex code. It may become more difficult to debug the font setup after the code is written.

Configuring the SDK

This section describes the third-party development target SDK installation and configuration. Specifically, the build file configuration as it relates to an SDK (for example, Scientific-Atlanta SDK).

The third-party SDK for a target platform is probably the most important component in defining a platform's build process. A typical SDK often provides source files, include files, compilation tools, compilation parameters, debug tools, and packaging tools that combine to enable building, executing, and debugging an application on a platform. The OCAP stack was designed to be executed on a platform as a standard application. Therefore, an important step in the process of enabling the OCAP stack on a platform involves identifying, extracting, and integrating from an SDK the elements of an application build process into the OCAP build environment framework.

SDK integration begins with installation of the SDK into the OCAP root directory. The standard location for platform SDKs is within the `tools` subdirectory of the OCAP root directory.



For detailed information on installing the latest SDK, refer to *Installing the SDK* section.

Typically, executable tools for building an application (for example, a compiler) are part of the SDK. Integrating these tools into the configuration file (`buildenv.bat`) is required. Use the following instructions to integrate the SDK-specific tools:

1. Open the `buildenv.bat` configuration file located within the platform's target configuration subdirectory. The subdirectory will depend on the type of build being created, debug or release. For example, `%OCAPROOT%\target\Vidion\%OCAPTC%\debug\buildenv.bat`.
2. Add the execution path for the SDK in the `buildenv.bat` using a DOS set command. For example, set
`PATH=%OCAPROOT%\tools\Win32\ABC\SDK_1.2.3\bin;%PATH%`

If the SDK has multiple subdirectories for its executable tools, then multiple path environment lines similar to the one above could be added to the associated `buildenv.bat` file.

Use the following instructions to integrate the various compiler tool commands, execution options, and parameters into the OCAP build files:

1. Review the example application makefiles from the SDK. The commands, options, and parameters specific to an SDK should be integrated into the `sdkrules.mak` file thus isolating these SDK specifics for a target platform. The `sdkrules.mak` file should be located in the `%OCAPROOT%\tools\Win32\operating system\SDK` directory.
2. Any command option or parameter that may need to vary for each target configuration is a good candidate for generalizing into the `buildrules.mak` file for the target configuration. There is typically a `buildrules.mak` file in both the debug and release in each `%OCAPROOT%\target\Vidion\%OCAPTC%` directories for a port. The `buildrules.mak` file will usually contain an `include` statement to pull in the SDK specific `sdkrules.mak` file contents, allowing for the sharing of common parameters with SDK specifics. For example, the following `buildrules.mak` excerpt shows some common makefile macro definitions with a final `sdkrules.mak` include statement:
 - ◆ `BUILD_CFLAGS = -g -DqDebug=1 -O1 -Wmissing-prototypes -Wstrict-prototypes`
 - ◆ `CONFIG_DEV = d`
 - ◆ `include $(PTV)/sdkrules.mak`

The `sdkrules.mak` file usually contains the bulk of the compilation specifics, which are typically extracted from an example SDK application makefile for the platform. It is not possible to list a complete set of requirements for what is defined within the `sdkrules.mak` file because the build process for platforms can vary greatly, but a general list of some common parameters would be:

- ◆ compiler library references: for example, `-l` options for the GCC compile
- ◆ compiler compilation flags: for example, GCC options: `-f`, `-w`, `-D`, etc.
- ◆ compiler pre-processor options: for example, GCC include options: `-I`, `-nostdinc`, etc.
- ◆ library generation options: for example, GCC: `cs`
- ◆ compilation command macros: for example, GCC library generation command: `ar`, etc.
- ◆ application packaging options for making a flashable application image: platform-specific to the target
- ◆ pre-compilation source processing commands and options: for example, execution of Perl scripts across sources for exposing function symbols for dynamic library support
- ◆ post-compilation image processing commands and options: for example, execution of Perl scripts across GCC produced `.a` files to generate new source files to expose function symbols for dynamic library support

3 Setting Up Your Environment

Overview

Porting the OpenCable Application Platform (OCAP) stack requires anticipating and adapting to the target platform's runtime environment. To minimize the need to recompile and redeploy the stack each time the target environment changes, Vidiom's design allows many target-specific accommodations to be deferred until the target platform is booted, at which time the OCAP stack can examine its environment and adapt accordingly.

This section provides configuration information for the following:

- ◆ `mpeenv.ini` file
- ◆ Signaling files
- ◆ Monitor Application

Before reading this chapter, you should be familiar with:

- ◆ OCAP stack runtime configuration
- ◆ the OCAP system constants as defined in the OCAP specification

After reading this chapter, you should be:

- ◆ able to identify and modify the files that may need to be configured for the target runtime environment

Startup configuration sequence

The OCAP stack configures itself to the runtime environment as the host device boots.

First, all environment variables are set as the OCAP stack reads and processes its initialization file, mpeenv.ini. The manner in which environment variables are “ingested” is platform-specific. Depending on the specific target platform being used, the mpeenv.ini initialization file can reside as a file on a disk, in flash memory, or can be built directly into the OCAP stack binary image.

Second, the OCAP stack reads and processes signaling files, such as hostapp.properties, and starts the host-resident applications. This mechanism is common to all ports of the Vidiom OCAP stack, and is not platform-specific.

Third, the OCAP stack attempts to read and process the XAIT, starting up a monitor application, if one was signaled in the XAIT. (Again, this mechanism is common to all ports of the Vidiom OCAP stack).

Configuring the mpeenv.ini file

The Multimedia Platform Environment (MPE) environment is configured through the `mpeenv.ini` initialization file, which contains environment variables that are initialized before the OCAP stack is loaded.



For detailed instructions on setting the `OCAPROOT` and `OCAPTC` environment variables, refer to the *Configuring the development system* section.

The SA-8300HD reference port uses the `mpeos_envGet()` method to read options from the initialization file at the beginning of the OCAP stack boot process.

The `mpeenv.ini` file is located in the `%OCAPROOT%\bin\Vidiom\%OCAPTC%` directory and is non-portable across platforms.

- ◆ **NOTE:** When setting environment variables, ensure there are no spaces before or after the equal sign (=).

The information included in the `mpeenv.ini` files can be divided into the following categories:

- ◆ *Port specific*
- ◆ *OCAP stack*
- ◆ *JVM specific*



For examples on how these environment variables are used, refer to the sample `mpeenv.ini` files in the sample port in the `%OCAPROOT%\bin\Vidiom\%OCAPTC%` debug or release directory.

Port specific

The following information is specific to a given port and needs to be customized for each platform during the porting process. The port-specific information includes fields defined in the following categories:

- ◆ *Debug*
- ◆ *Display*
- ◆ *File system*
- ◆ *Font*
- ◆ *Front panel*
- ◆ *Graphic*
- ◆ *Manager*
- ◆ *Memory*
- ◆ *Network*
- ◆ *OCAP extensions*
- ◆ *Security*
- ◆ *System commands*
- ◆ *System host*
- ◆ *User interface*

Debug

Use the debug, port-specific fields to configure the debugging options. Example debugging fields are:

INVOKE.DEBUG.PRE.JVM=switch

determines whether the code drops into the debugger (for example, GDB) before the Java Virtual Machine (JVM) is enabled. Where:

switch determines whether the pre-JVM debug option is enabled. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default.

For example:

INVOKE.DEBUG.PRE.JVM=FALSE

INVOKE.DEBUG.PRE.MAIN=switch

determines whether the code drops into the debugger (for example, GDB) before the OCAP Java main entry point is called. Where:

switch determines whether main code debug is enabled.

Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default.

For example:

INVOKE.DEBUG.PRE.MAIN=FALSE

INVOKE.DEBUG.PRE.MPE=switch

determines whether the code drops into the debugger (for example, GDB) before MPE is initialized. Where:

switch determines whether the pre-MPE debug option is enabled. Possible values for *switch* are:

TRUE enables the pre-MPE debug option.

FALSE disables the option. FALSE is the default.

For example:

INVOKE.DEBUG.PRE.MPE=FALSE

INVOKE.JVM=switch

determines whether the JVM is enabled. Where:

switch determines whether the JVM is enabled. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default.

For example:

INVOKE.JVM=FALSE

LOG.MPE.module=level

determines the module and level of logging. Where:

module specifies the logical area of code to set the logging level. Possible values for *module* are:

DEFAULT

specifies all the modules.

CC specifies the closed captioning module.

CDL specifies the common download module.

COND specifies the MPE synchronization module, which has APIs to create mutexes and condition variables.

DBG specifies the debug module.

DISP specifies the display module.

DLL specifies the Dynamic Link Library (DLL) management module.

DVR specifies the Digital Video Recorder (DVR) module.

EVENT specifies the event module.

ED specifies the event dispatch module.

FILESYS

specifies the file system module.

FILTER specifies the section filtering module.

FP specifies the front panel module.

GFX specifies the graphics module.

JAVA specifies the Java code module.

JNI specifies the Java Native Interface (JNI) module.

JVM specifies the JVM module.

MEDIA specifies the media module.

MEM specifies the memory module.

MUTEX specifies the thread synchronization module.

NET specifies the network module.

OS specifies the operating system module.

POD specifies the Point-Of-Deployment (POD) module.

SI specifies the service information module.

SOUND specifies the sound module.

STORAGE specifies the storage module.

SYS specifies the system manager module.

TARGET specifies miscellaneous platform-specific logging.

TEST specifies the MPE testing framework module.

THREAD specifies the thread module.

UTIL specifies the utility module.

UI specifies the user interface module.

level specifies the logging level for the associated module. Possible values for *level* are:

- ALL specifies FATAL, ERROR, WARN, and INFO messages are logged.
- FATAL specifies fatal error messages are logged.
- ERROR specifies regular error messages are logged.
- WARN specifies warning messages are logged.
- INFO specifies informational messages are logged.
- DEBUG specifies debug messages are logged.
- TRACE*n* specifies a tracing level messages are logged. *n* specifies the level of tracing (1-9).

For example:

```
LOG.MPE.DEFAULT=ALL
```

Log entries are passed to a separate utility for parsing, so the syntax for log entries differs from that of other parameters. An exclamation point (!) negates a value, and for some *module* values, multiple *level* values may be permitted. For example:

```
LOG.MPE.FILTER = ALL !INFO !DEBUG
```

MPE.BOOTMODE=*mode*

specifies whether fast-boot mode is enabled. Where:

mode determines whether boot mode is set for development or production. Possible values for *mode* are:

fast specifies development boot mode. fast is the default setting.

normal specifies production boot mode.

For example:

```
MPE.BOOTMODE=fast
```

TEST.MPE=*switch*

determines whether the low-level MPE/Multimedia Platform Environment Operating System (MPEOS) test harness is enabled. Where:

switch determines whether test harness is enabled. Possible values for *switch* are:

TRUE enables the option. TRUE is the default.

FALSE disables the option.

For example:

```
TEST.MPE=TRUE
```

Display

Use the display, port-specific fields to configure the display options. Example display fields are:

DEFAULT.VDM.FORMAT=*format*

determines the default output format for the video display manager. Where:

format specifies the output format. Possible values for *format* are:

1 specifies YPrPb/YPbPr analog color space.
1 is the default setting.

3 specifies Digital Visual Interface (DVI).

For example:

```
DEFAULT.VDM.FORMAT=1
```

DEFAULT.VDM.MODE=*resolution*

determines the default screen resolution output for the video display manager. Where:

resolution

specifies the resolution settings. Possible values for *resolution* are:

0 specifies 480i standard definition. 0 is the default setting.

2 specifies 480p enhanced definition.

3 specifies 720p high definition.

4 specifies 1080i high definition.

For example:

```
DEFAULT.VDM.MODE=4
```

DEFAULT.VDM.PREF=zoom

determines the default zoom mode output for the video display manager. Where:

zoom specifies the default zoom setting. Possible values for *zoom* are:

- 0 specifies letterboxing or wide-screen, masking-off areas above and below the picture area. 0 is the default setting.
- 1 specifies pan and scan or full screen, cropping the sides of the image to fit the screen.
- 3 specifies distorted, stretching the picture and eliminating the original aspect ratio.

For example:

DEFAULT.VDM.PREF=0

File system

Use the file system port-specific fields to configure the OCAP file system parameters at the MPE level. Example file-system fields are:

FS.DEFSYSDIR=/fileSystem

defines the default file system. Where:

fileSystem

specifies the path to the default file system directory.

For example:

FS.DEFSYSDIR=/romfs

FS.ROMFS.LIBDIR=libraryDirectory

defines the file system library directory. Where:

libraryDirectory

specifies the path to the default library directory.

For example:

FS.ROMFS.LIBDIR=snfs:/

OCAP.persistent.appstorage=path

overrides the default path that will be used to store downloaded OCAP applications. The default path is defined by the OCAP.persistent.root environment variable, plus /app. Where:

path specifies the path used to define the location for OCAP application storage.

For example:

OCAP.persistent.appstorage=/itfs/app

`OCAP.persistent.dvbroot=path`

overrides the default path that will be used to define the OCAP property dvb.persistent.root. The default path is defined by the `OCAP.persistent.root` environment variable, plus /usr. Where:

path specifies the path used to define the `dvb.persistent.root` OCAP property.

For example:

`OCAP.persistent.dvbroot=/itfs/usr`

`OCAP.persistent.dvr=path`

overrides the default path that will store DVR metadata files. The default path is defined by the `OCAP.persistent.root` environment variable, plus /dvr. Where:

path specifies the full path for storage of DVR metadata.

For example:

`OCAP.persistent.dvr=/itfs/dvr`

`OCAP.persistent.root=path`

defines the base directory for persistent storage on the platform. Where:

path specifies the root directory for persistent storage of files.

For example:

`OCAP.persistent.root=/itfs`

`OCAP.persistent.userprefs=path`

overrides the default path that will be used to store OCAP user preferences. The default path is defined by the `OCAP.persistent.root` environment variable, plus /prefs. Where:

path specifies the path used to store OCAP user preferences.

For example:

`OCAP.persistent.userprefs=/itfs/prefs`

`VMDLLPATH=path`

defines the path to the JVM DLL library module. Where:

path specifies the path to the JVM DLL library module.

For example:

`VMDLLPATH=snfs://evm/bin/evm.ptv`

Font

Use the font, port-specific fields to configure the built-in system fonts. Example font fields are:

SYSFONT=fontName[,fontName,[...]]

uniquely defines the system fonts. Where:

fontName

specifies the logical name of the font.

For example:

SYSFONT=arial,tiresias

-
- ◆ **NOTE:** Always define the mandatory OCAP font Tiresias as the last font listed in SYSFONT.
-

**SYSFONT .fontName[.style][.n]=fontFile ,
[minSize[-maxSize]]**

uniquely defines a specific associated system font and attributes. Where:

fontName

specifies the logical name of the font, as specified in the SYSFONT field.

style specifies the optional style type. Possible values for *style* are:

PLAIN specifies that the font is not bold and not italic. PLAIN is the default.

BOLD specifies a bold font.

ITALIC

specifies the font is italicized.

BOLD_ITALIC

specifies a bold and italic font.

n

defines an optional identifier. There are five maximum identifiers ranging from 0 to 4. Use the identifier when you have two or more *fontName* or *fontName* plus *style* definitions. If no identifier is defined, the identifier is 0.

fontFile

specifies the absolute path to the font file.

minSize

specifies the optional minimum font size in points for the logical font to support. If the font size is defined, *minSize* is required. If *minSize* is not defined, the font sizes defined in the font file are used.

maxSize

specifies the optional maximum font size in points.

For example:

```
SYSFONT.Tiresias.BOLD.1=/romfs/sys/fonts/
Tires-o_802.pfr,10-48
```

Front panel

Use the front panel, port-specific fields to configure the front panel commands used for host devices that support a font-panel text display. Example front panel command fields are:

OCAP.api.option.fp=version

specifies the OCAP optional API support for the front panel as defined in OCAP 13.3.12.2. Applications can detect the presence of a front panel implementation by checking for the definition of the Java system property `ocap.api.option.fp`.

Where:

version specifies the OCAP version. The default value is 1.0.

For example:

```
OCAP.api.option.fp=1.0
```

OCAP.fp.characters=characters

determines the character set supported by the text display on the front panel. Where:

character

determines character set. The default value is

ABCDEFGHIJLNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz-_.

For example:

```
OCAP.fp.characters=ABCDEFGHIJLNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz-_
```

OCAP.fp.indicators=list

displays on an OCAP interface a space-separated list of indicator names. Each host may support front-panel indicators that are not intended to be controlled by OCAP applications. The MPEOS implementation for each platform defines and detects the complete list of indicators supported by a particular host. This property was added to allow limiting of the actual list of indicators available to OCAP applications to a subset of all the supported indicators. Where:

list specifies a space-separated list of indicator names (listed in the front-panel MPEOS driver) that are visible from the OCAP interface.

For example:

```
OCAP.fp.indicators=message text
```

Graphic

Use the graphic, port-specific fields to configure the unsupported graphic commands. An example graphic command field is:

`DFBDLLPATH=fileSystem :path`

specifies the DirectFB DLL. DirectFB is a complete hardware abstraction layer with software fallbacks for every graphics operation that is not supported by the underlying hardware. Where:

fileSystem

specifies the name of file system.

path specifies the absolute path to the DirectFB DLL.

For example:

`DFBDLLPATH=s nfs://directfb.ptv`

Manager

Create manager, port-specific fields to override the manager property options. Example manager fields are:

`OCAP.mgrmgr.Resource=resourceManager`

specifies what class will be instantiated to act as the `com.vidiom.impl.ResourceManager` and provide the `org.ocap.resource.ResourceContentionManager` functionality.

`com.vidiom.impl.manager.resource.RecordingRezMgr` provides functionality specific to DVR-enabled systems - such as shared resource usages and resource contention warnings.

For example:

`OCAP.mgrmgr.Resource=com.vidiom.impl.manager.resource.RecordingRezMgr`

`OCAP.mgrmgr.ResourceReclamation=reclaimManager`

specifies an override for the default resource reclamation manager as defined in the `mgrmgr.properties` resource file. `mgrmgr.properties` is located in the `ocap-rez.jar` as `com/vidiom/impl/manager/mgrmgr.properties`.

For example:

`OCAP.mgrmgr.ResourceReclamation=com.vidiom.impl.manager.reclaim.EvmRRMgr`

`OCAP.mgrmgr.Service=serviceManager`
 specifies the class
`com.vidiom.impl.manager.service.DVRServiceMgrImpl`
 which provides necessary support for recorded services and
 is required for a configuration that supports the recording
 and playback of recorded services. To effectively override
 the default
`com.vidiom.impl.manager.service.ServiceMgrImpl` class,
 include an entry like the following in the `mpeenv.ini` file:
`OCAP.mgrmgr.Service=com.vidiom.impl.manager.`
`service.DVRServiceMgrImpl`

-
- ◆ **NOTE:** This extension of the `ServiceMgrImpl` class supports both broadcast and recorded services.
-

`OCAP.mgrmgr.Storage=storageManager`
 specifies an override for the current `storageManager` settings
 in the `mgrmgr.properties` resource file. `mgrmgr.properties`
 is located in the `ocap-rez.jar` as
`com/vidiom/impl/manager/mgrmgr.properties`.

For example:

`OCAP.mgrmgr.Storage=com.vidiom.impl.manager.`
`service.DVRStorageMgrImpl`

Memory

Use the memory, port-specific fields to configure the memory options.
 Example memory fields are:

`heap.n=size[measurement]`

maps to the operating system heap. This is the heap that is used as a fallback if another heap cannot fulfill a request. This heap is used, by default, for all colors. Adding memory heap definitions is optional. The default setting is `3m`. Where:

`n` specifies an increasing integer number. Assign each heap a unique `heap.n` number, starting with 0. The heaps are used in successive order starting with `heap.0` until all the assigned heaps are used. Possible values range from 0 to 6.

`size` specifies the allocated size of the heap in either kilobytes or megabytes.

`measurement`

specifies the measurement of the size. Possible values for `measurement` are:

`k` specifies kilobytes.

`m` specifies megabytes.

For example:

`heap.1=3m`

`heap.n.colors=type[,type]`

specifies the type of memory to allocated. Where:

n specifies a memory type a unique `heap.n.colors` number, starting with 1. The types of memory are allocated in successive order starting with `heap.1.color` until all the memory types are allocated. Possible values range from 1 to 7.

Use these unique numbers to limit the amount of memory that can be allocated for that type(s), and to reduce fragmentation between allocations of this type and others.

type specifies the color/type of memory to be included in the heap. Memory should be grouped in like types. Once grouped, the memory areas are combined and treated as if they are the same memory. Currently, the types are defined in `mpe_MemColor` that is defined in `mpe_types.h`. The following is a brief description of `mpe_MemColor`:

`mpe_MemColor`

identifies the type of memory to include in the heap. Add new ones as necessary. Generally, there should be at least one type for every logical code module, with potentially additional sub-areas per code module. You should keep the total number under 32 total to allow use in bit-field. When adding a color, be sure to also update `mpe/os/all targets/mpeos_mem.c` with a new string for the color. `mpe_MemColor` is defined as follows:

```
typedef enum {
    MPE_MEM_SYSTEM = -1,
    MPE_MEM_GENERAL = 0,
    MPE_MEM_TEMP,
    MPE_MEM_JVM,
    MPE_MEM_GFX,
    MPE_MEM_GFX_LL,
    MPE_MEM_GFX_FONT,
    MPE_MEM_SYNC,
    MPE_MEM_THREAD,
    MPE_MEM_FILE,
    MPE_MEM_FILE_CAROUSEL,
    MPE_MEM_MEDIA,
    MPE_MEM_UTIL,
    MPE_MEM_EVENT,
    MPE_MEM_FILTER,
    MPE_MEM_POD,
    MPE_MEM_SI,
    MPE_MEM_TEST,
```

```

MPE_MEM_NET,
MPE_MEM_SECTION_FILTER,
MPE_MEM_TABLE_FILTER,
MPE_MEM_OPAQUE_TABLE,
MPE_MEM_OPAQUE_FILTER,
MPE_MEM_CC,
MPE_MEM_DVR,
MPE_MEM_NCOLORS
} mpe_MemColor;

```

Where:

- MPE_MEM_SYSTEM
specifies GetFreeSize and GetLargestFree only.
- MPE_MEM_GENERAL
specifies a generic color/type of memory.
- MPE_MEM_TEMPspecifies temporary memory.
- MPE_MEM_JVM
specifies JVM that includes system and Java heaps memory.
- MPE_MEM_GFX
specifies generic graphics memory.
- MPE_MEM_GFX_LL
specifies low-level graphics memory (for example, DFB).
- MPE_MEM_GFX_FONT
specifies font memory (for example, FT2).
- MPE_MEM_SYNC
specifies synchronization primitives memory.
- MPE_MEM_THREAD
specifies threads.
- MPE_MEM_FILE
specifies generic file system memory.
- MPE_MEM_FILE_CAROUSEL
specifies data and object carousel memory.
- MPE_MEM_MEDIA
specifies generic media memory.
- MPE_MEM_UTILspecifies utility memory.
- MPE_MEM_EVENT
specifies event memory.
- MPE_MEM_FILTER
specifies section filtering memory.

MPE_MEM_POD
specifies CableCard/POD memory.

MPE_MEM_SI
specifies service information memory.

MPE_MEM_TEST
specifies memory used during testing.

MPE_MEM_NET
specifies networking memory.

MPE_MEM_SECTION_FILTER
specifies section filtering memory.

MPE_MEM_TABLE_FILTER
specifies table filtering memory.

MPE_MEM_OPAQUE_TABLE
specifies opaque table memory.

MPE_MEM_OPAQUE_FILTER
specifies opaque filter memory.

MPE_MEM_CC
specifies closed captioning module memory.

MPE_MEM_DVR
specifies DVR manager module memory.

MPE_MEM_NCOLORS
specifies all types of memory.

For example:

`heap.1.colors=4`

heap.n.fallback=[switch]
specifies whether a heap fallback is used. Where:

n specifies the heap number.

switch determines whether the heap fallback is enabled.
Possible values for *switch* are:

0 enables the option.

1 disables the option. 1 is the default.

For example:

`heap.0.fallback=1`

MEM.SYS.POLL.INTERVALS=seconds

specifies the number of seconds between polling intervals for heap size changes. Where:

seconds specifies the number of seconds to wait. The default setting is 30.

For example:

`MEM.SYS.POLL.INTERVALS=2`

MEM.SYS.RECLAIM.THRESH=*kilobytes*

monitors the size of the heap for resource depletion. If memory falls below the designated number, the system attempts to reclaim 1.5 times that the number of kilobytes.

Where:

kilobytes

specifies the number of kilobytes.

For example:

```
MEM.SYS.RECLAIM.THRESH=0x80000
```

MEM.SYS.RESTRICT.SIZE=*kilobytes*

specifies the amount of memory available after a soft power-up. The size is only an approximation, and used only for debugging. Where:

kilobytes

specifies the number of kilobytes. If the size is not available (either to small or to big), it defaults to 1024KB.

For example:

```
MEM.SYS.RESTRICT.SIZE=1024
```

Network

Use the network, port-specific fields to configure the network, manager-property options. Example network fields are:

OCAP.rcInterface.dataRate=*rate*

defines the data rate. Where:

rate specifies the rate that data is passed. Possible values for *rate* are:

1544 specifies a rate of 1.544 Mbps. This is the only supported setting.

For example:

```
OCAP.rcInterface.dataRate=1544
```

OCAP.rcInterface.subType=*carousel*

defines the carousel. Where:

carousel

specifies the type of carousel. Possible values for *carousel* are:

SUBTYPE_CATV_00B

specifies an out-of-band carousel. Currently, this is the only supported setting.

For example:

```
OCAP.rcInterface.subType=SUBTYPE_CATV_00B
```

OCAP extensions Use the OCAP extensions, port-specific fields to configure the OCAP extension packages that expose additional APIs to OCAP application Xlets. An example OCAP extension field is:

`OCAP.extensions=extension[,extension,[,...]]`

specifies optional OCAP extension API packages. Where:

extension

specifies an extension package to the OCAP stack.
Possible values for *extension* are:

`com.vidiom.debug`

specifies the proprietary Vidiom debug
extension package.

For example:

`OCAP.extensions=com.vidiom.debug`

Security

Use the security, port-specific fields to configure the security options.
Example security fields are:

MPE.ROOTCERTS=*path*

specifies the location of the Root Certificate used for file authentication in the OCAP stack. Where:

path specifies path for the Root Certificate. For this reference platform, the location is assumed to be a path relative to the RomFS file system (/romfs/).

For example:

MPE_.ROOTCERTS=sys/certs/CLCVCROOT.perm

-
- ◆ **NOTE:** The example indicates the Root Certificate is located at the /romfs/sys/certs/CLCVCROOT.perm.
-

OCAP.security.policy.*i*=*permission*

implicitly grant signed host device application additional privileges. Where:

i specifies an increasing integer number. Assign each permission a unique OCAP.security.policy.*n* number, starting with 0. The security policies are used in successive order starting with OCAP.security.policy.0 until all the assigned policies are used. By default, possible values range from 0 to 31, this may be overridden by defining the OCAP.security.policy.*n* variable.

permission

specifies the permission to give to the signed host application. Permissions should be specified using the following format *permission* := *type*[*name*[*actions*]]:

type specifies a fully-qualified classname of permission (for example, java.io.FilePermission).

name specifies a string with no spaces, as would be given as a name argument to java.security.Permission or java.security.BasicPermission constructors.

action(s)

specifies the action or actions separated by commas, as would be given to the java.security.BasicPermission constructor.

The following example syntax grants all signed host applications read and write permission to files located under /snfs:

```
OCAP.security.policy.0=
    java.io.FilePermission /snfs/- read,write
```

System commands Use the system command, port-specific fields to configure the system commands. System commands are run before the MPE/OCAP stack is brought up these are processed in contiguous ascending order, beginning with SYSCMDS.0. An example system-command field is:

- ◆ **NOTE:** System commands are used only in a development environment because a release build does not have a shell available to run commands.

SYSCMDS.n=systemCommand

contains system commands (shell commands) that run before the MPE/OCAP stack is initialized. Where:

n specifies an increasing integer number. Assign each system command to a unique SYSCMDS.*n* number, starting with 0. There is no limit to the number of system commands you can assign. The system commands are run in successive order starting with SYSCMDS.0 until all the assigned commands are run.

systemCommand

specifies the exact string sent to the system. Do not use quotes unless they are to be passed in with the command itself. System command strings are very port and operating system dependent, and are most likely non-portable across platforms.

For example:

```
SYSCMDS.0=log set all off
SYSCMDS.1=log set ld info
SYSCMDS.2=gdb_set enable 1
SYSCMDS.3=log set dma severe
```

System host

Use the system-host settings to configure platform-specific capabilities. Example system-host settings are:

MPE.SYS.ACOUTLET=switch

specifies whether the host device supports application control of an AC outlet. Where:

switch determines whether the AC outlet is controllable. Possible values for *switch* are:

TRUE indicates the host device supports control of an AC outlet. TRUE is the default.

FALSE indicates the host device does not support control of an AC outlet.

For example:

MPE.SYS.ACOUTLET=TRUE

MPE.SYS.RFBYPASS=switch

specifies whether the host device supports application control of a RF-bypass. Where:

switch determines whether the RF bypass is controllable. Possible values for *switch* are:

TRUE indicates the host device supports control of the RF bypass.

FALSE indicates the host device does not support control of the RF bypass. FALSE is the default.

For example:

MPE.SYS.RFBYPASS=FALSE

User interface

Use the user interface, port-specific field to configure the user interface. An example user-interface field is:

DEFAULT.UI.EVENT.QUEUE.SIZE=size

defines the maximum size of events for the default user-interface (for example, key input events) event queue. Where:

size specifies the queue size in megabytes. Possible values range from 0 to 64. The default setting is 64.

For example:

DEFAULT.UI.EVENT.QUEUE.SIZE=64

OCAP stack

The following information is specific to the OCAP stack and should not be changed for a given platform unless you understand all of the ramifications throughout the OCAP stack at both the Java and MPE/MPEOS layers. The OCAP stack information includes fields defined in the following categories:

- ◆ *Miscellaneous*
- ◆ *DVR*
- ◆ *Emergency Alert System (EAS)*
- ◆ *Host*
- ◆ *Manager*
- ◆ *MPE*
- ◆ *Object carousel*
- ◆ *Security*
- ◆ *Service information*
- ◆ *Signaling*
- ◆ *Time*

Miscellaneous

Use these miscellaneous, stack-specific fields in a development environment to configure different interfaces. The miscellaneous field is:

`OCAP.prf.I16=switch`

determines whether the OCAP stack is backwards compatible with the Permission-Request File (PRF) defined in OCAP 1.0 I16 (for example: `ocap:monitorapplication name="registeredapi"`). Where:

`switch` specifies whether the PRF is supported. Possible values for `switch` are:

`TRUE` enables PRF support. `TRUE` is the default.

`FALSE` disables PRF support.

For example:

`OCAP.prf.I16=TRUE`

DVR

Use the DVR, stack-specific fields to configure the DVR interface. The DVR fields are:

DEFAULT.POWER.STATE=*state*

determines the default state of the DVR device. Where:

state determines the state. Possible values for *state* are:

0 disables the option.

1 enables the on state.

2 enables the standby state. 2 is the default.

For example:

DEFAULT.POWER.STATE=2

DVR.LOW.POWER.RESUME.TIME=*milliseconds*

determines how many milliseconds the slowest DVR device takes to resume from a low-power mode. Where:

milliseconds

specifies the number of milliseconds. Set *milliseconds* to 0 to disable resume mode. 120000 is the default.

For example:

DVR.LOW.POWER.RESUME.TIME=120000

DVR.ON.LOW.POWER.IDLE.TIME=*minutes*

determines how many minutes the DVR device remains idle in the on mode before going into the low-power mode.

Where:

minutes

specifies the number of minutes. To disable low-power mode in the on state, set *minutes* to 0. The default setting is 0.

For example:

DVR.ON.LOW.POWER.IDLE.TIME=0

DVR.STANDBY.LOW.POWER.IDLE.TIME=*minutes*

determines how many minutes the DVR device remains idle in the standby mode before going into the low-power mode. Where:

minutes

specifies the number of minutes. To disable low-power mode in the standby state, set *minutes* to 0. The default setting is 0.

For example:

DVR.STANDBY.LOW.POWER.IDLE.TIME=0

DVR.TSB.EXTRA.SPACE=space

determine how much extra space in bytes is required when a Time-Shift Buffer (TSB) is created. The TSB creation will fail if the available space is not at least equal to the TSB requested size + that extra space. Where:

space specifies the amount of extra space in bytes needed in the TSB. 616038400 is the default.

For example:

```
DVR.TSB.EXTRA.SPACE=616038400
```

-
- ◆ **NOTE:** If you want to create a TSB of 616038400 bytes and the setting is DVR.TSB.EXTRA.SPACE=616038400, the total required space will be double this number.
-

MPE.TARGET.DISABLE.UNSESS=switch

determines whether sess_DisableUNSession() is called when MPE starts. Where:

switch determines whether sess_DisableUNSession() is called. Possible values for *switch* are:

TRUE enables the option. TRUE is the default.

FALSE disables the option.

For example:

```
MPE.TARGET.DISABLE.UNSESS=TRUE
```

ocap.api.option.dvr=version

specifies the version of the DVR extension.

ocap.api.option.dvr is a system property required by the OCAP Specification. ocap.api.option.dvr is not an option and is used by the applications to discover the version of the DVR API implemented in the OCAP stack. Where:

version determines the version of the DVR API implementation. Changing the value has no effect on the API version and should be left at the default value (1.0) in the mpeenv.ini file.

For example:

```
ocap.api.option.dvr=1.0
```

OCAP.dvr.recording.leaveOrphans=switch

determines whether the recording manager saves or deletes recordings for which it has no metadata. By default, if the recording manager discovers recordings not associated with any known scheduling information ("orphaned" recordings), it will remove those recordings from the hard drive.

switch determines whether to save or delete the orphaned recordings. Possible values for switch are:

TRUE removes the recordings for which it has no scheduling information. TRUE is the default setting.

FALSE leaves the recordings.

For example:

OCAP.dvr.recording.leaveOrphans=TRUE

OCAP.dvr.ts.bufferingAutostartDelay=*milliseconds*

determines the amount of time a DVR enabled ServiceContext waits after tuning to start buffering.

OCAP.dvr.ts.bufferingAutostartDelay is used only if buffering is enabled. Where:

milliseconds

specifies the number of milliseconds (for example, 2000 = 2 seconds). The default setting is 2000.

For example:

OCAP.dvr.ts.bufferingAutostartDelay=2000

Emergency Alert System (EAS)

Use the EAS, stack-specific fields to configure the EAS interface. The EAS fields are:

OCAP.eas.ccTimeout=*milliseconds*

determines the amount of time to allow all contexts to be invoked. Where:

milliseconds

specifies the number of milliseconds. The default setting is 5000.

For example:

OCAP.eas.ccTimeout=5000

`OCAF.eas.listenerTimeout=milliseconds`

determines the amount of time between when all contexts have been invoked and before continuing with EAS processing. Where:

milliseconds

specifies the number of milliseconds. The default setting is 5000.

For example:

`OCAF.eas.listenerTimeout=5000`

`OCAF.eas.tune.interrupt.appId=AppID`

specifies the HostEASKeyListenerXlet AppID. Where:

AppID specifies the actual application identifier as defined in the hostapp.properties. The default setting is 0x0000003B0013.

For example:

`OCAF.eas.tune.interrupt.appId=0x0000003B0013`

Host

Use the host, stack-specific fields to configure the host interface. The host fields are:

`MPE.SYS.ID=identifier`

identifies the unique ID string for this host device. This could be the RF MAC address (which is unique between manufacturers). `MPE.SYS.ID=identifier` is used by the org.ocap.hardware.Host.getID() method. Where:

identifier

specifies the host identifier.

For example:

`MPE.SYS.ID=112233445566`

- ◆ **NOTE:** Most ports define this environment variable at run-time instead of through the mpeenv.ini file.

MPE.SYS.RFMACADDR=*address*

identifies the MAC address for the host's RF reverse-unicast, communication channel. This value matches the value the host uses in DOCSIS Set-top Gateway (DSG) mode for a Dynamic-host Configuration Protocol (DHCP) request. MPE.SYS.RFMACADDR=*address* is used by the org.ocap.hardware.Host.getReverseChannelMAC() method. Where:

address identifies the MAC address consisting of 12 hexadecimal characters. The *address* string must be in 123456789ABC format, not in 12:34:56:78:9A:BC format.

For example:

MPE.SYS.RFMACADDR=123456789ABC

Manager

Use the manager, stack-specific fields to configure the authentication manager. The manager fields are:

OCAP.mgrmgr.Auth=*authenticationManager*

specifies the authentication manager used for validating signed files. Possible values for *authenticationManager* are:

com.vidiom.impl.manager.auth.AuthManagerImpl

validates the files are properly signed before allowing access to the file.

com.vidiom.impl.manager.auth.NoAuthentication

bypasses authentication, always allowing any access to any file regardless of whether it is properly signed.

For example:

OCAP.mgrmgr.Auth=com.vidiom.impl.manager.auth.
AuthManagerImpl

MPE

Use the MPE, stack-specific fields to configure the MPE interface. The MPE field is:

MainClassArgs.n

When defining the main (initial) virtual machine class and parameters, argument zero identifies the class containing a static main method, and is not passed to the main function as the other arguments are OCAP entry points. The MainClassArgs.n is passed to the MPE JVM Manager to determine the main class to execute and the arguments to use. Changing the main class will affect whether the OCAP stack runs or not. Specifying a different main class can be used to run a standalone Java application or even unit tests. Where:

n specifies an increasing integer number. Assign each argument a unique MainClassArgs.n number, starting with 0. There is no limit to the number of arguments you can assign. The arguments are passed in successive order starting with MainClassArgs.0 until all the assigned arguments are run.

For example:

```
MainClassArgs.0=com.vidiom.impl.ocap.OcapMain
```

The first argument (MainClassArgs.0) specifies the main OCAP implementation. The second argument (MainClassArgs.1) specifies an optional implementation. Once the last option has been set, the next listing in the mpeenv.ini file is evaluated.

Object carousel

Use the object carousel, stack-specific fields to configure the object carousel interface. The object carousel fields are:

OC.CHECK.VERSION=switch

determines whether the filter that monitors changes in the Dynamic Invocation Interface (DII) and version changes in files is enabled. Where:

switch determines whether the monitoring filter is enabled. Possible values for *switch* are:

TRUE enables the option. TRUE is the default.

FALSE disables the option.

For example:

```
OC.CHECK.VERSION=TRUE
```

OC.CHECK.VERSION.OOB=*switch*

determines whether monitoring for Dynamic Invocation Interface (DII) version changes for Out-Of-Band (OOB) carousels is enabled. Where:

switch determines whether the monitoring is enabled.

Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. If set to FALSE, `DSMCCObject.setObjectChangeListener()` may never (or very rarely) signal that the versions have changed. FALSE is the default.

For example:

```
OC.CHECK.VERSION.OOB=FALSE
```

OC.DECOMPRESSION.SIZE=*bytes*

determines the number of bytes to decompress at one time when accessing compressed modules. Larger values are typically faster, but use more memory. Where:

bytes specifies the number in bytes to compress. Possible values range from 0 to 32768. The default setting is 32768.

For example:

```
OC.DECOMPRESSION.SIZE=32768
```

OC.DEFAULT.TIMEOUT=*milliseconds*

specifies the default time-out timer for DII and Download Server Indication (DSI). Where:

milliseconds

specifies the number of milliseconds of the timer. Possible values range from 0 to 300000. The default setting is 300000.

For example:

```
OC.DEFAULT.TIMEOUT=300000
```

OC.DII.CHECK.DURATION=*milliseconds*

specifies the amount of time to leave a Download Information Indication (DII) check filter in place before removing it, unless more accesses occur. Where:

milliseconds

specifies the number of milliseconds of the timer. Possible values range from 0 to 10000. The default setting is 10000.

For example:

```
OC.DII.CHECK.DURATION=10000
```

OC.ENABLE.BIOP.CACHE=*switch*

determines whether the parsed Broadcast Inter-Object Request Broker (ORB) Protocol (BIOP) objects are cached. BIOP objects are BIOP objects are the headers for files in the object carousel. Enabling OC.ENABLE.BIOP.CACHE provides a very specific, very small type of cache that can potentially make large performance improvement at a small cost in memory. Where:

switch determines whether the parsed BIOP objects are cached. Possible values for *switch* are:

TRUE enables caching. TRUE is the default.

FALSE disables caching.

For example:

```
OC.ENABLE.BIOP.CACHE=TRUE
```

OC.MAX.CACHESIZE=*size*

determines the amount of memory in bytes for the entire object carousel cache (the file data). Where:

size specifies the size in bytes of the object carousel cache.

For example:

```
OC.MAX.CACHESIZE=3584000
```

OC.MAX.PREFETCH=*n*

specifies the maximum number of pre-fetched, total, to allow outstanding. If less than pre-fetch start, this will override pre-fetch start. Where:

n specifies the total number of pre-fetched allowed. Possible values range from 0 to 8. The default setting is 8.

For example:

```
OC.MAX.PREFETCH=8
```

OC.PREFETCH.DISTANCE=*n*

specifies how far ahead to look for sections to pre-fetch when searching for pre-fetched. Where:

n specifies the number of pre-fetched to look ahead. Possible values range from 0 to 8. The default setting is 8.

For example:

```
OC.PREFETCH.DISTANCE=8
```

`OC.PREFETCH MODULES=switch`

specifies how the module is pre-fetch. Where:

`switch` specifies pre-fetcheds on entire modules or blocks within modules. Possible values for `switch` are:

`TRUE` specifies the object carousel will attempt to pre-fetch entire modules when they are first accessed. `TRUE` is the default.

`FALSE` specifies only the blocks will be fetched when accessed, with some additional prefetching as indicated by other `OC.` environment variables.

For example:

`OC.PREFETCH MODULES=TRUE`

`OC.PREFETCH MODULESIZE=size`

specifies the maximum size of a module that is pre-fetched. Where:

`size` specifies the total size of the module. If the module size is smaller than the `size` specified, the OCAP stack will attempt to pre-fetch the entire carousel when the carousel is mounted, or updated (via a change in the DII). If the total size is greater than this, it will not be pre-fetched, instead it will be accessed in the same demand fetch manner we use now (including any per-module/per-file pre-fetching which is already implemented).

-
- ◆ **NOTE:** The size includes all overhead in the module, not just the total size of data in the module.
-

For example:

`OC.PREFETCH MODULESIZE=65536`

`OC_PREFETCH_MOUNTSIZE=size`

specifies a threshold size. Where:

`size` specifies the total size of the carousel. If the carousel size is smaller than the `size` specified, the stack will attempt to prefetch the entire carousel when the carousel is mounted, or updated (via a change in the DII). If the total size is greater than this, it will not be prefetched, instead it will be accessed in the same demand fetch manner we use now (including any per-module/per-file prefetching which is already implemented).

For example:

`OC.PREFETCH MOUNTSIZE=8388608`

OC.PREFETCH.MOPUP=*switch*

determines, after completion of prefetching an entire module, if all the blocks were successfully prefetched. If any of the Downloadable Data Blocks (DDB) are missing, OC.PREFETCH.MOPUP issues a separate prefetch request for those individual blocks. Where:

switch determines whether the option is enabled. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default setting.

For example:

```
OC.PREFETCH.MOPUP=FALSE
```

OC.PREFETCH.OOB=*switch*

determines whether the OOB object carousel attempts to do prefetches. Where:

switch determines whether the option is enabled. Possible values for *switch* are:

TRUE enables the option. TRUE is the default setting.

FALSE disables the option.

For example:

```
OC.PREFETCH.OOB=TRUE
```

OC.PREFETCH.START=*n*

specifies how many pre-fetches to start on an object carousel cache miss. Where:

n specifies the number of pre-fetches. Possible values range from 0 to MAX_INT. In practice, the maximum value of OC.PREFETCH.START is constrained to the value of OC.MAX.PREFETCH. If set to 0, pre-fetching is disabled. The default setting is 4.

For example:

```
OC.PREFETCH.START=4
```

OC.USE.DVB.NSAP=*type*

specifies the type of format for the Network Service Access Point (NSAP) addresses. Where:

type specifies the type of addresses. Possible values for *type* are:

FALSE specifies OCAP format NSAP addresses.
FALSE is the default.

TRUE specifies DVB format NSAP addresses.

For example:

```
OC.USE.DVB.NSAP=FALSE
```

Service information Use the service information, stack-specific fields to configure the service information interface. The service information fields are:

SITP.PSI.DISABLED.OOB.PSI=*switch*

controls only the acquisition of Program Specific Information (PSI) on the OOB channel for either extended channel (CableCard/POD or DSG broadcasting). Both extended channels are supported for debugging and demo purposes. Where:

switch determines whether PSI acquisition on the OOB channel is enabled. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default setting.

For example:

```
SITP.PSI.DISABLE.OOB.PSI=FALSE
```

SITP.PSI.ENABLED=*switch*

controls all PSI acquisition on available MPEG transport streams, primarily, Program Association Table (PAT) and Program Map Table (PMT). Where:

switch determines whether PSI processing is enabled.

Possible values for *switch* are:

TRUE enables the option. TRUE is the default setting.

FALSE disables the option.

For example:

```
SITP.PSI.ENABLED=TRUE
```

`SITP.PSI.FILTER.NUMBER=n`

specifies the number of filters to use for PSI acquisition.
`SITP_PSI_ENABLED` must be enabled. Where:

n determines the number of filters. The default is 1.

For example:

`SITP.PSI.FILTER.NUMBER=1`

- w **NOTE:** Currently, this values is hard coded as 1 in the `sitp_psi.c`.
-

`SITP.PSI.OOB.FILTERS.ALWAYS.SET=switch`

optimizes PSI thread to round-robin OOB PSI filters.
 Round-robin is a simple time-sharing method which allows each resource user to use the resource for a particular duration. Where:

switch determines whether optimizing PSI thread is enabled. Possible values for *switch* are:

`TRUE` enables the option. `TRUE` is the default setting.

`FALSE` disables the option.

For example:

`SITP.PSI.OOB.FILTERS.ALWAYS.SET=TRUE`

`SITP.PSI.PROCESS.OOB.TABLE.REVISIONS=switch`

enables monitoring for changes in the OOB PAT and the PMT. Where:

switch determines whether Out-Of-Band (OOB) monitoring is enabled. Possible values for *switch* are:

`TRUE` enables the option. `TRUE` is the default setting.

`FALSE` disables the option.

For example:

`SITP.PSI.PROCESS.OOB.TABLE.REVISIONS=TRUE`

SITP.PSI.PROCESS.TABLE.REVISIONS=switch
 enables monitoring for changes in the in-band PAT/PMT.
 Where:

switch determines whether in-band monitoring is enabled.
 Possible values for *switch* are:

TRUE enables the option. TRUE is the default setting.

FALSE disables the option.

For example:

SITP.PSI.PROCESS.TABLE.REVISIONS=TRUE

SITP.PSI.RETRY.INTERVAL=*millisecond*
 specifies the period of time in milliseconds between retries to the in-band (PAT/PMT) filters on failure. Where:

milliseconds

specifies the number of milliseconds between retries on failure. Possible values for *millisecond* range from 0 to 20000. The default is 2000.

For example:

SITP.PSI.RETRY.INTERVAL=2000

SITP.PSI.ROUND.ROBIN.INTERVAL=*millisecond*
 specifies the period of time in milliseconds to wait for a PSI table version change. Where:

milliseconds

specifies the number of milliseconds to wait. Possible values for *millisecond* range from 0 to 2000. The default is 2000.

For example:

SITP.PSI.ROUND.ROBIN.INTERVAL=2000

SITP.SI.DUMP.CHANNEL.TABLES=switch
 specifies the Network Information Table (NIT)/Short-form Virtual Channel Table (SVCT)/Network Text Table (NTT) entries are dumped to an output during initial acquisition or a new revision is acquired. This feature is used only in the debug build and is independent of logging flag settings. Where:

switch determines whether the table entries are deleted.
 Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default setting.

For example:

SITP.SI.DUMP.CHANNEL.TABLES=FALSE

SITP.SI.ENABLED=switch

specifies out-of-band service information processing.

Where:

switch determines whether service information processing is enabled. Possible values for *switch* are:

TRUE enables the option. TRUE is the default setting.

FALSE disables the option.

For example:

```
SITP.SI.ENABLED=TRUE
```

SITP.SI.FILTER.MULTIPLIER=n

specifies the number of multiplier used when setting filters to collect multi-section tables. Since no table can be guaranteed to be signaled and acquired contiguously from first to last section, *n* is multiplied against the number of sections in the table to guarantee that all sections will be acquired in the subsequent section acquisition round.

Where:

n determines the number of multipliers used. Possible values for *n* range from 2 to 10. The greater the number the higher the impact of table processing during initial acquisition and revisioning. The default is 2.

For example:

```
SITP.SI.FILTER.MULTIPLIER=2
```

SITP.SI.MAX.SECTION.SEEN.COUNT=n

specifies the number of counts used as a comparator to determine if the section Cyclical Redundancy Check (CRC) repetition is high enough to confidently assert, using CRC revisioning, that all table sections have been found for any table. Where:

n determines the number of counts used. Possible values for *n* range from 0 to 10. The greater the number the higher the impact of table processing during initial acquisition and revisioning. The default is 4.

For example:

```
SITP.SI.MAX.SECTION.SEEN.COUNT=4
```

SITP.SI.MAX.UPDATE.POLL.INTERVAL=*millisecond*

specifies the maximum period of time in milliseconds for polling OOB tables (NIT/SVCT/NTT) after the initial acquisition. The polling window interval is a random value between SITP.SI.MIN.UPDATE.POLL.INTERVAL and SITP.SI.MAX.UPDATE.POLL.INTERVAL. Where:

milliseconds

specifies the maximum number of milliseconds.

Possible values for *millisecond* range from 0 to 30000.

The default is 30000.

For example:

```
SITP.SI.MAX.UPDATE.POLL.INTERVAL=30000
```

SITP.SI.MIN.UPDATE.POLL.INTERVAL=*millisecond*

specifies the minimum period of time in milliseconds for polling OOB tables (NIT/SVCT/NTT) after the initial acquisition. The polling window interval is a random value between SITP.SI.MIN.UPDATE.POLL.INTERVAL and SITP.SI.MAX.UPDATE.POLL.INTERVAL. Where:

milliseconds

specifies the minimum number of milliseconds.

Possible values for *millisecond* range from 0 to 30000.

The default is 25000.

For example:

```
SITP.SI.MIN.UPDATE.POLL.INTERVAL=25000
```

SITP.SI.PROCESS.TABLE.REVISIONS=*switch*

specifies monitoring for changes in the OOB NIT/SVCT. Where:

switch determines whether monitoring is enabled. Possible values for *switch* are:

TRUE enables the option. TRUE is the default setting.

FALSE disables the option.

For example:

```
SITP.SI.PROCESS.TABLE.REVISIONS=TRUE
```

`SITP.SI.REV.SAMPLE.SECTIONS=switch`

specifies revision table acquisition using the a sampling size equal to the initial table size. Where:

`switch` determines whether revision table acquisition is enabled. Possible values for `switch` are:

`TRUE` enables the option.

`FALSE` disables the option. `FALSE` is the default setting.

For example:

`SITP.SI.REV.SAMPLE.SECTIONS=FALSE`

`SITP.SI.STATUS.UPDATE.TIME.INTERVAL=milliseconds`

specifies the time in milliseconds before the SIDB service information data state is set to SI_NOT_AVAILABLE_YET, releasing early service information acquisition lock in JavaTV. Where:

`milliseconds`

specifies the minimum number of milliseconds.

Possible values for `milliseconds` range from 0 to 15000.

The default is 15000.

For example:

`SITP.SI.STATUS.UPDATE.TIME.INTERVAL=15000`

`SITP.SI.VERSION.BY.CRC=switch`

specifies versioning for SCTE 65 Specification, Profile 1 compliance by the Cyclical Redundancy Check (CRC) for service information. Where:

`switch` determines whether versioning is enabled. If the table does not have a Revision Detection Descriptor (RDD), the default is 1. Possible values for `switch` are:

`FALSE` disables the option. `FALSE` is the default setting.

`TRUE` enables the option.

For example:

`SITP.SI.VERSION.BY.CRC=FALSE`

Signaling

Use the signaling, stack-specific fields in a development environment to configure the signaling interface. The signaling fields are:

`OCAP.ait.ignore=switch`

determines if the Application Information Table (AIT) is ignored or acknowledged. Where:

switch specifies whether AIT is ignored or acknowledged.
Possible values for *switch* are:

TRUE acknowledges the option.

FALSE ignores the option. FALSE is the default.

For example:

`OCAP.ait.ignore=FALSE`

`OCAP.xait.ignore=switch`

determines if the Extended Application Information Table (XAIT) is ignored or acknowledged. Where:

switch specifies whether XAIT is ignored or acknowledged.
Possible values for *switch* are:

TRUE acknowledges the option.

FALSE ignores the option. FALSE is the default.

For example:

`OCAP.xait.ignore=FALSE`

`OCAP.xait.I15=switch`

determines whether interpretation of XAIT-specific descriptors as specified in issued release 15 (I15) or 16 (I16) of the OpenCable Application Platform Specification in addition to the latest supported version of the specification. Where:

switch specifies whether OCAP 1.0 I15 and I16 XAIT descriptor tags are supported. Possible values for *switch* are:

TRUE indicates that I15- and I16-compliant descriptor tags are recognized in addition to the descriptor tags described in the latest supported specification. TRUE is the default.

FALSE indicates that I15- and I16-compliant descriptor tags are not supported.

For example:

`OCAP.xait.I15=TRUE`

- ◆ **NOTE:** Signaling commands are used only in a development environment.

| | |
|---------------------|--|
| Time | <p>Use the time, stack-specific field in a development environment to configure the time interface. The time field is:</p> <p><code>OCAP.timer.maxtardiness=milliseconds</code></p> <p>specifies the maximum tardiness of a timer task. If a timer task is tardy, a system time change may have taken place. In this case, the timer task will be rescheduled to its next execution time in the future based on the new system time. The default setting is 300000. Where:</p> <p><i>milliseconds</i></p> <p>specifies the maximum tardiness of a timer task. Possible values for <i>milliseconds</i> range from 0 to 300000. The default is 300000.</p> <p>For example:</p> <pre>OCAP.timer.maxtardiness=300000</pre> |
| JVM specific | <p>The following information is specific to a particular JVM and should not be changed unless you are using a different JVM or you understand all of the ramifications to the specific JVM that is being used in this port. The examples included in this section are based on the OCAP stack using the Esmertec Virtual Machine (EVM). If you are using a different virtual machine, these fields will need to be reconfigured.</p> |
| Format | <p>The format of the JVM environment variables is:</p> <p><code>VMOPT.n=property=value</code></p> <p>contains the virtual machine options (VMOPT.) that run before the MPE/OCAP stack is initialized. Where:</p> <p><i>n</i> specifies an increasing integer number. Assign each option a unique VMOPT.<i>n</i> number, starting with 0. VMOPT.COUNT is used to limit to the number of options you can assign. The virtual machine options are run in successive order starting with VMOPT.0 until all the assigned options are run.</p> <p><i>property</i> specifies the Java property to assign a value. Properties prefaced with -D are used for general purposes and properties prefaced with -X are used for debugging purposes.</p> <p><i>value</i> specifies the value to set the property.</p> <p>For example:</p> <pre>VMOPT.1=-Djava.library.path=snfs:/</pre> |

`VMOPT.COUNT=max_count`

specifies the maximum number of Virtual Machine (VM) options. Possible values are:

max_count

specifies an integer number. The default is 32, which means that possible values for *n* in the `VMOPT.n` list range from 0 to 31. If additional entries are required, override this default limit by specifying a larger value for *count*.

Properties and values

Use the following property and value, JVM-specific fields to configure your virtual machine options:

`-Djava.class.path=path`

specifies a semicolon-separated system class path where OCAP stack classes are found. Where:

path specifies the absolute path to the system class paths.

For example:

```
VMOPT.0=-Djava.class.path=ocap.icb;/romfs/sys/ocap-rez.jar;/romfs/sys/support.jar;/romfs/sys/sciatldia.g.jar;/romfs/qa/xlet;/romfs/sys/OemAppsExt.jar;/romfs/sys/ProfileExt.jar;/romfs/sys/DebugExt.jar;/romfs/sys/podh.jar;/romfs/usr/ocapExt.jar;/romfs/sys/dvupgrade.jar;/romfs/sys/HDDManager.jar;/romfs/odn;/romfs/ptvapps;/romfs/apps;/romfs/qa
```

-
- ◆ **NOTE:** `ocap.icb` specifies a OnAlready jar file, whose contents are located in a DLL (for PowerTV the file is `ocap.ptv`). This defines a Java system property.
-

`-Djava.compiler=NONE`

determines if the Dynamic Adaptive Compiler that compiles bytecodes to native instructions at runtime is enabled. It is recommended that the compiler be disabled when debugging. This variable is currently commented out.

For example:

```
VMOPT.15=-Djava.compiler=NONE
```

`-Djava.home=library`

defines `java.home` system property. This is used to locate libraries. Where:

library specifies the virtual machine.

For example:

```
VMOPT.3=-Djava.home=evm
```

`-Djava.library.path=fileSystem`
specify semicolon-separated native library path. Where:

fileSystem
specifies the path to the native libraries.

For example:

```
VMOPT.1=-Djava.library.path=snfs:/
```

-
- ◆ **NOTE:** You must specify the file system using native file naming, which is port specific. This defines a Java system property.
-

`-Djava-vm.name=name`
defines the name of the virtual machine. Where:

name specifies the name of the virtual machine.

For example:

```
VMOPT.18=-Djava-vm.name=evm
```

`-Djeode.evm.compiler.codeBuffTotal=memory`
specifies the maximum space in bytes for holding the dynamically-compiled code of your application. Where:

memory
specifies maximum space for holding the dynamically-compiled code of your application, in bytes, as a decimal number optionally followed by a **k** or an **m** for a scaling of 1024 or 1024 * 1024 respectively. The space actually used will depend on your application. The default setting is 1000k.

For example:

```
VMOPT.9=-Djeode.evm.compiler.codeBuffTotal=500k
```

-
- ◆ **NOTE:** Compaction implies a non-concurrent garbage collection cycle.
-

`-Djeode.evm.compiler.managerThread=switch`
specifies whether threads are created to manage code buffers. Where:

switch determines whether threads are created to manage code buffers. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option and reduces the number of threads used. **FALSE** is the default setting.

For example:

```
VMOPT.8=-Djeode.evm.compiler.managerThread=FALSE
```

-Djeode.evm.console.remote.remoteJava=*switch*

sends Java output to remote console. Commented out by default. Where:

switch determines whether Java output is sent to a remote console. Possible values for *switch* are:

TRUE sends Java output to a remote console.

FALSE the option is disabled.

For example:

VMOPT.31=-Djeode.evm.console.remote.remoteJava=TRUE

-Djeode.evm.console.remote.remoteNative=*switch*

sends native virtual machine output to remote console. Commented out by default. Where:

switch determines whether native virtual machine output is sent to a remote console. Possible values for *switch* are:

TRUE sends virtual machine output to a remote console.

FALSE disables the option.

For example:

VMOPT.30=-Djeode.evm.console.remote.remoteNative=TRUE

-Djeode.evm.feedback.enable=*switch*

determines if the JeodeMonitor tool is enabled. Where:

switch determines whether JeodeMonitor tool is used. This variable is currently commented out. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option.

For example:

VMOPT.13=-Djeode.evm.feedback.enable=TRUE

-Djeode.evm.feedback.suspend=*switch*

determines if the virtual machine is suspended until JeodeMonitor tool connects. This variable is currently commented out. Where:

switch determines whether the virtual machine is suspended. Possible values are:

TRUE suspends the virtual machine.

FALSE enables the virtual machine.

For example:

VMOPT.14=-Djeode.evm.feedback.suspend=TRUE

`-Djeode.evm.memory.gc.epochSpan=milliseconds`

determines the time in milliseconds between garbage collection cycles. Where:

milliseconds

specifies the number of milliseconds between garbage collection cycles. Time is measured from the start of one cycle to the start of the next cycle. If set to 1, Jbed cycles as often as possible. If set to 0, Jbed never performs a cycle unless memory actually becomes exhausted or a cycle is explicitly requested by `Runtime.gc()`. The default value is 20000.

For example:

`VMOPT.10=-Djeode.evm.memory.gc.epochSpan=20000`

`-Djeode.evm.memory.gc.infoLevel=level`

controls the level of detailed information provided in verbose garbage collection output. Where:

level specifies the level of information to provide. The default value is 0x17f.

For example:

`VMOPT.7=-Djeode.evm.memory.gc.infoLevel=0x13f`

`-Djeode.evm.memory.gc.maxPriority=priority`

sets the maximum priority of the garbage collection thread. Where:

priority

specifies the priority level. Possible values for *n* range from 0 to 10. The default is 1.

For example:

`VMOPT.12=-Djeode.evm.memory.gc.maxPriority=1`

`-Djeode.evm.memory.javaExtend=size`

provides information to the VM to ensure that approximately this much memory is available without having to grow the heap again. Commented out by default. Where:

size specifies the size of the heap in bytes, as a decimal number optionally followed by a **k** for a scaling of 1024. The default setting is 256k.

For example:

`VMOPT.17=-Djeode.evm.memory.javaExtend=256k`

`-Djeode.evm.memory.javaStart=size`

specifies the initial size of the Java heap allocated by the EVM at startup. Where:

size specifies the size of the heap in bytes, as a decimal number optionally followed by a **k** or an **m** for a scaling of 1024 or 1024 * 1024 respectively. Values are determined based on your port. The default setting is 7m.

For example:

```
VMOPT.16=-Djeode.evm.memory.javaStart=7m
```

`-verbose` or `-verbose:arguments`

enables verbose virtual machine execution. Where:

arguments

specifies a comma-separated list of argument.

Possible values for *arguments* are:

`gc` reports on each garbage collection event.

`class` information about each class loaded is displayed.

`jni` reports native methods and other Java Native Interface activity.

If no arguments are defined, `gc` and `class` are implied.

For example:

```
VMOPT.2=-verbose:class,jni,gc
```

`-Xdebug`

enables debugging. Commented out by default.

For example:

```
VMOPT.29=-Xdebug
```

`-Xlinenum`

enables line numbering in exception OCAP stack traces.

For example:

```
VMOPT.4=-Xlinenum
```

`-Xmxsize`

sets limit on the Java object heap, in bytes, as a decimal number optionally followed by a **k** or an **m** for a scaling of 1024 or 1024 * 1024 respectively. For example, the following entry sets a 7 Megabyte (MB) limit on the Java object heap:

```
VMOPT.5=-Xmx7m
```

`-Xnoverify`

disables the JVM's bytecode verifier which speeds up class load time.

For example:

```
VMOPT.6=-Xnoverify
```

-Xrunjdwp:address=8000,server=y,suspend=y
instructs the VM to suspend operation until the debugger attaches. You must enable debugging (Xdebug). Commented out by default.

For example:

VMOPT.28=-Xrunjdwp:address=8000,server=y,suspend=y

Signaling files

The OCAP implementation supports an alternate form of application signaling through Java properties files. This format is always used to describe resident host device manufacturer



An AIT is used to signal in-band applications and an XAIT is used to signal out-of-band applications in the development environment.

applications. This format can also be used to represent in-band application signaling (AIT) and out-of-band (OOB) Multiple Systems Operator (MSO) application signaling (XAIT). The following Java properties file are used to configure the application signaling:

- ◆ The `ait-xxxx.properties` file can be used as an alternative way to configure bound applications. Bound applications are typically associated with a service (for example, ABC, CBS, and NBC) where normally the signaling is done through the AIT.
- ◆ The `xait.properties` file can be used as an alternative way to configure unbound applications. Unbound applications (for example, the Electronic Programming Guide (EPG), Video On Demand (VOD), Pay-Per-View (PPV)) are MSO network specific. Normally, the signaling is done through the XAIT.
- ◆ The `hostapp.properties` file is used to configure the applications that are resident on the host device (for example, a manufacturer-provided application).

The `SignallingManager` implementation determines whether normal network-based application or `.properties`-based application signaling is used. The `DavicSignallingMgr` implementation (`com.vidiom.impl.manager.signalling` package), that supports true OCAP application signaling, is meant to be the default. However, the `TestSignallingMgr` implementation (`com.vidiom.impl.manager.signalling.TestSignallingMgr` package), that support `.properties`-based signaling can be used by defining the following system property:

```
-Dmgrmgr.Signalling=com.vidiom.impl.manager.signalling.TestSignallingMgr
```

In general, all ports should use the `DavicSignallingMgr` package. However, this is currently not the case for Windows (`TestSignallingMgr`) or PowerTV (`PowerTvSignallingMgr`). In most cases, new ports will use `TestSignallingMgr` to control signaling during implementation and testing, and move to Digital Audio Video Council (DAVIC) section filtering when it is available.

Configuring the ait-xxxx.properties file

The AIT form of signaling is used to describe in-band application signaling during development. As long as a service is selected, the implementation will watch for changes to the ait-xxxx.properties file. Where xxxx is the service identifier expressed in the following form: [1-9a-f][0-9a-f] (lowercase hexadecimal, with no leading zeros). The ait-xxxx.properties file is located along the system class-path. The AIT types of fields used for signaling are:



For additional details on AIT fields, refer to the Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification.

- ◆ *Version*
- ◆ *Transport protocols*
- ◆ *Applications*
- ◆ *External authorization*

-
- ◆ **NOTE:** ait-xxxx.properties example properties file is an AIT signaling emulation via a properties file and is used only in a development environment.
-

Version

Use the version field to identify the application signaling version for AIT signaling. If no definition is provided, then the implicit entry `version=0` is assumed. The version field uses the following format:

`version=n` determines the version of the application signaling. Where:
`n` specifies the version level. `version` is a 5-bit field able to hold 0-31 decimal values.

-
- ◆ **NOTE:** Signaling updates are acknowledged only with a version change.
-

Transport protocols

Use the transport-protocol fields to configure transport protocols for AIT signaling. By default, if no transport protocols are defined, an implicit local protocol is defined as `transport.255=local`.

- ◆ **NOTE:** The local transport always has a label of 255, regardless of the index used in the definition. For example, the app.i.transport must reference 255 to get the local transport.

The transport fields have the following format:

`transport.n=protocol`

determines the transport protocol. Where:

n specifies an increasing integer number. Assign each transport a unique `transport.n` number, starting with 0. Possible values range from 0 to 31. To redefine the range of values, define a property named `maxtp` and assign it a value other than the default 32.

`protocol`

specifies the type of protocol. Possible values for `protocol` are:

`oc` specifies the object carousel protocol.

`ip` specifies Internet protocol.

`local` specifies a local protocol.

`transport.n.key=value`

defines the value of the transport-stream identifier. Where:

n specifies an increasing integer number. Assign each transport a unique `transport.n` number, starting with 0. Possible values range from 0 to 31. To redefine the range of values, use `maxtp`.

`key` specifies the variables for that transport stream protocol definition. Possible values for `key` are:

`remote=switch`

determines whether the protocol is remote. This is a required field when the type is `oc` or `ip`. Possible values for `switch` are:

`TRUE` enables the option.

`FALSE` disables the option.

`service=identifier`

specifies the abstract service identifier, but only if `remote=TRUE`. This is a required field when the type is `oc` or `ip`. For example, `0x41a`.

component=n

identifies the principal service component that delivers the application. This is a required field when the protocol is oc. component is the 8-bit component tag specifying the elementary stream that carries the DS1 of the object carousel.

alignment=switch

determine whether the alignment indicator is enabled. This is a required field when the type is ip. Possible values for switch are:

TRUE enables the option.

FALSE disables the option.

url.n specifies an increasing integer number.

Assign each URL a unique url.n number, starting with 0. Possible values range from 0 to 31. To redefine the number of URLs, use maxurl.

value specifies the value associated with the specified key.

Applications

Use the following format to configure the application fields:

app.n.field=value

specifies the application, the application's unique information, and the value associated with the information. Where:

n specifies an increasing integer number. Assign each application a unique app.n number, starting with 0. The initial range of values is from 0 to 31. You can run up to this many applications within the limited space of the set-top box. To redefine the range of values, define a property called maxapps and assign it a value other than the default 32. The applications are run in successive order starting with app.0 until all the assigned applications are run. Assign each transport a unique transport.n number, starting with 0.

field specifies information about your set-top box applications.

value specifies the value associated with the specified field.

| | |
|--------------------------------|--|
| Application identifiers | Application identifier ranges are defined for AIT-signaled applications, for XAIT-signaled applications, and for manufacturer-defined host applications. For the <code>ait-xxxx.properties</code> file, which simulates an OCAP/DVB AIT, valid application identifier ranges are: |
| | 0x0000-0x3fff specifies unsigned applications. |
| | 0x4000-0x7fff specifies signed applications. |
| Required fields | Use the required application fields to define how and when an application gets signaled. The required application fields are: |
| | <code>app.n.application_control_code=mode</code> determines how to control the application. Where: |
| | <i>n</i> specifies a unique <code>app.n</code> number, starting with 0. The initial range of values is from 0 to 31. |
| | <i>mode</i> specifies how to control the application. Possible values for <i>mode</i> are: |
| | PRESENT starts the application control now. |
| | AUTOSTART automatically starts the application control when the set-top box is booted. |
| | KILL stops the application control. |
| | REMOTE designates the remote control for application control. |
| | <code>app.n.application_identifier=identifier</code> uniquely identifies the application and its affiliated organization. The identifier value is used to keep track of running applications. Where: |
| | <i>n</i> specifies a unique <code>app.n</code> number, starting with 0. The initial range of values is from 0 to 31. |
| | <i>identifier</i> uniquely identifies the application and its affiliated organization. The <code>application_identifier</code> is comprised of a 32-bit unique organization identifier (OID) and a 16-bit application identifier (AID). For example, 0x000000017001: OID=0x00000001, AID=0x7001. The application identifier determines the base level of permissions granted the application. Possible values for <i>identifier</i> are: |
| | 0x0000-0x3fff specifies unsigned application permissions. |
| | 0x4000-0x7fff specifies signed permissions (plus Permission Request File (PRF)) if desired. |

`app.n.base_directory=path`

specifies the path to the application. Where:

n specifies a unique app.*n* number, starting with 0.
The initial range of values is from 0 to 31.

path specifies the Unix-style absolute path to the application.

- ◆ **NOTE:** The base_directory path in the properties signaling can be absolute or relative. If relative, the path is considered to be relative to the defined system default directory. This is a port-dependent concept. On Win32 it corresponds to the current working directory, on other platforms it corresponds to the FS.DEFSYSDIR system property. This applies only if application.0.transport is undefined or defined as application.0.transport=255 and transport.255=local is defined.

`app.n.initial_class_name=name`

defines the initial class name of the application implementation. Where:

n specifies a unique app.*n* number, starting with 0.
The initial range of values is from 0 to 31.

name specifies the initial class name (com.yourcompany.apps.YourXlet) of the application implementation.

`app.n.priority=priority`

assigns resources to the set-top box. Higher priority applications get a larger share of available resources, such as Central Processing Unit (CPU) time and memory. Where:

n specifies a unique app.*n* number, starting with 0.
The initial range of values is from 0 to 31.

priority specifies the resource priority of the application.
Possible values for *priority* are:

255 specifies the highest priority. 255 is used by the Monitor Application.

100 to 254
specifies unbound applications.

1 to 200
specifies bound applications.

`app.n.visibility=visibility`

controls whether the application is visible to the viewer.
Where:

n specifies a unique app.*n* number, starting with 0.
The initial range of values is from 0 to 31.

visibility

specifies if the application is visible to the viewer.
Possible values for *visibility* are:

INVISIBLE

specifies the application is not visible.

VISIBLE

specifies the application is visible.

VISIBLE-TO-APPS-ONLY

specifies the applications can only be used by other applications.

Optional fields

Use the optional application fields to configure your application options.
Optional application fields are:

app.n.application_name=name

assigns the name of the application. Where:

n specifies a unique app.*n* number, starting with 0.
The initial range of values is from 0 to 31.

name specifies the name associated with the application.

app.n.application_name.argumentNumber=language ,name

assigns the language- specific name expressed as *lang ,name*.
Where:

n specifies a unique app.*n* number, starting with 0.
The initial range of values is from 0 to 31.

argumentNumber

tracks the number of arguments in an increasing integer value.

language

specifies the language associated with the application.

name specifies the name associated with the application.

app.n.args.argumentNumber=argument=setting

Use the argument fields to initialize application parameters.
Arguments are passed through the properties file. Where:

n specifies an increasing integer number. Assign each application a unique app.*n* number, starting with 0. By default, up to 32 unique applications may be defined. You can run up to this many applications within the limited space of the set-top box. The applications are run in successive order starting with app.0 until all the assigned applications are run.

argumentNumber

specifies an increasing integer number. Assign each argument a unique *args.n* number, starting with 0. The arguments are run in successive order starting with *args.0* until all the arguments are run. By default, up to eight unique arguments may be defined.

argument

specifies the argument The argument is defined by each of the application separately.

setting specifies the value of the argument.

app.n.classpath_extension=path

specifies classpath extension directories. Where:

n specifies a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31.

path specifies a semi-colon separated list of classpath extension directories.

app.n.icon_flags=path

specifies the icon flags as appropriate for AppIcon. Where:

n specifies a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31.

path specifies the Unix-style absolute path to the icon flags.

app.n.icon_locator=path

specifies the relative path to the base_directory for icons. Where:

n specifies a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31.

path specifies the Unix-style absolute path to the icon locator.

app.n.service_bound=switch

determines whether the application is service bound or can continue to live with the next service. An application is always part of a service, but if not set, service is ignored. If set to TRUE, the application dies when the viewer selects away from that service. If set to FALSE, the application lives until it is determined that the application can live in the next service. Where:

n specifies a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31.

switch specifies a Boolean to indicate if the application is service bound. Possible values for *switch* are:

TRUE enables the option. TRUE is the default.

FALSE disables the option.

app.n.transport=number

specifies the transport protocol indices. Where:

n specifies a unique *app.n* number, starting with 0.
The initial range of values is from 0 to 31.

number

specifies a comma-separated list of transport protocols defined in *transport.n*.

External authorization

Use the external authorization fields to configure your application identifier and the priority. The external authorization entries are used in AIT signaling.

- ◆ **NOTE:** External authorization is not needed if the application is signaled through the *app.n.application_identifier* field.

The external authorization fields use the following format:

authorized.n=appld:priority

authorizes an external application and assigns a priority.
Where:

n specifies an increasing integer number. Assign each external application a unique *authorized.n* number, starting with 0. Possible values range from 0 to 31. To redefine the range of values, define a property named *maxauth* and assign it a value other than the default 32.

appld uniquely identifies the external application. The identifier value is used to keep track of running applications. The *appld* is comprised of a 32-bit unique organization identifier (OID) and a 16-bit application identifier (AID). For example, 0x000000017001: OID=0x00000001, AID=0x7001. The application identifier determines the type of permissions that the application may have. Possible values for *appld* are:

0x0000-0x3fff

specifies unsigned-application permissions.

0x4000-0x7fff

specifies signed-application permissions (plus additional permissions out of the application's PRF).

priority specifies the resource priority of the application.

Possible values for *priority* are:

255 specifies the highest priority. 255 is used by the Monitor Application.

100 to 254
specifies unbound applications.

1 to 200
specifies bound applications.

Example file

The following is an example ait-xxxx.properties file:

```
version=1

transport.0=oc
transport.0.component=6

app.1.application_identifier=0xcafed00d4d01
app.1.application_control_code=AUTOSTART
app.1.visibility=VISIBLE
app.1.priority=200
app.1.application_name.0=eng,Launcher
app.1.service_bound=false
app.1.initial_class_name=com.ajax.xlet.launcher.LauncherXlet
app.1.base_directory=/app/InbandLauncher
app.1.transport=0
app.1.args.0=showMenu

app.2.application_identifier=0xcafed00d4d02
app.2.application_control_code=PRESENT
app.2.visibility=VISIBLE
app.2.priority=180
app.2.application_name.0=eng,Checkers
app.2.initial_class_name=com.ajax.xlet.games.checkers.CheckersXlet
app.2.base_directory=/app/Checkers
app.2.classpath_extension=/lib/ui;/lib/games
app.2.transport=0
app.2.args.0=player=red

authorized.0=0x000000010001:22
authorized:1=0xcafebabe4001:100
```

Configuring the xait.properties file

The XAIT form of signaling can be enabled to describe MSO abstract services and their applications during development. If you decide to enable the XAIT form of signaling, you will need to create the xait.properties file and put it along the system class-path. You can create an implementation that can watch for changes to an xait.properties file instead of changes to network-signaled XAIT changes visible via the POD OOB interaction channel. The XAIT types of fields used for signaling are:



For specific details on XAIT property files, refer to the OpenCable Application Platform Specification.

- ◆ *Version*
- ◆ *Transport protocols*
- ◆ *Applications*
- ◆ *Services*

-
- ◆ **NOTE:** xait.properties is used only in a development environment.
-

Version

Use the version field to identify the application signaling version for XAIT signaling. If no definition is provided, then the implicit entry `version=0` is assumed. The version field uses the following format:

`version=n` determines the version of the application signaling. Where:
`n` specifies the version level. `version` is a 5-bit field able to hold 0-31 decimal values.

-
- ◆ **NOTE:** Signaling updates are acknowledged only with a version change.
-

Transport protocols

Use the transport protocol fields to configure transport protocols for XAIT signaling. By default, if no transport protocols are defined, an implicit local protocol is defined as `transport.255=local`.

-
- ◆ **NOTE:** The local transport always has a label of 255, regardless of the index used in the definition. For example, the `app.i.transport` must reference 255 to get the local transport.
-

The transport fields have the following format:

`transport.n@protocol`
determines the transport protocol. Where:
`n` specifies an increasing integer number. Assign each transport a unique `transport.n` number, starting with 0. Possible values range from 0 to 31. To redefine the range of values, use `maxtp`.

protocol

specifies the type of protocol. Possible values for *protocol* are:

oc specifies the object carousel protocol.

ip specifies Internet protocol.

local specifies a local protocol.

transport.n.key=value

defines the value of the transport-stream identifier. Where:

n specifies an increasing integer number. Assign each transport a unique *transport.n* number, starting with 0. Possible values range from 0 to 31. To redefine the range of values, use *maxtp*.

key specifies the variables for that transport stream protocol definition. Possible values for *key* are:

remote=switch

determines whether the protocol is remote. This is a required field when the type is **oc** or **ip**. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option.

service=identifier

specifies the abstract service identifier, but only if *remote=TRUE*. This is a required field when the type is **oc** or **ip**.

component=n

identifies the principal service component that delivers the application. This is a required field when the protocol is **oc**. *component* is the 8-bit component tag specifying the elementary stream that carries the DS1 of the object carousel.

alignment=switch

determine whether the alignment indicator is enabled. This is a required field when the type is **ip**. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option.

url.n specifies an increasing integer number. Assign each URL a unique *url.n* number, starting with 0. Possible values range from 0 to 31. To redefine the number of URLs, create a property named *maxurl* and assign it a value other than the default 32.

value specifies the value associated with the specified *key*.

Applications

Use the following format to configure the application fields:

`app.n.field=value`

specifies the application, the application's unique information, and the value associated with the information.
Where:

n specifies an increasing integer number. Assign each application a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31. You can run up to that many applications within the limited space of the set-top box. To redefine the range of values, use *maxapps*. The applications are run in successive order starting with *app.0* until all the assigned applications are run. Assign each transport a unique *transport.n* number, starting with 0.

field specifies information about your set-top box applications.

value specifies the value associated with the specified *field*.

Application identifiers

Application identifier ranges are defined for AIT-signaled applications, for XAIT-signaled applications, and for manufacturer-defined host applications. For the *xait.properties* file, which simulates an OCAP XAIT, valid application identifier ranges are:

0x0000-0x3fff
specifies unsigned applications.

0x4000-0x5fff
specifies signed applications.

0x6000-0x7fff
specifies dual-signed applications.

Required fields

Use the required application fields to define how and when an application is to be signaled. The required application fields are:

`app.n.application_control_code=mode`

determines how to control the application. Where:

n specifies a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31.

mode specifies how to control the application. Possible values for *mode* are:

PRESENT

starts the application control now.

AUTOSTART

automatically starts the application control when the set-top box is booted.

KILL stops the application control.

REMOTE designates the remote control for application control.

`app.n.application_identifier=identifier`

uniquely identifies the application and its affiliated organization. The identifier value is used to keep track of running applications. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

identifier

uniquely identifies the application and its affiliated organization. The `application_identifier` is comprised of a 32-bit unique organization identifier (OID) and a 16-bit application identifier (AID). For example, 0x000000017001: OID=0x00000001, AID=0x7001. The application identifier determines the base level of permissions granted the application. Possible values for *identifier* are:

0x0000-0x3fff

specifies unsigned-application permissions.

0x4000-0x5fff

specifies signed-application permissions (plus additional permissions out of the application's PRF).

0x6000-0x7fff

specifies Monitor Application permissions (plus additional permissions out of the application's PRF).

`app.n.base_directory=path`

specifies the path to the application. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

path specifies the Unix-style absolute path to the application.

- ◆ **NOTE:** The `base_directory` path in the properties signaling can be absolute or relative. If relative, the path is considered to be relative to the defined system default directory. This is a port-dependent concept. On Win32 it corresponds to the current working directory, on other platforms it corresponds to the `FS.DEFSYSDIR` system property. This applies only if `application.0.transport` is undefined or defined as `application.0.transport=255` and `transport.255=local` is defined.

`app.n.initial_class_name=name`

defines the initial class name of the application implementation. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

name specifies the initial class name
 (com.yourcompany.apps.YourXlet) of the application implementation.

app.n.priority=priority
 assigns resources to the set-top box. Higher priority applications get a larger share of available resources, such as CPU time and memory. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

priority specifies the resource priority of the application. Possible values for *priority* are:

255 specifies the highest priority. 255 is used by the Monitor Application.

100 to 254 specifies unbound applications.

1 to 200 specifies bound applications.

app.n.service=identifier

identifies the abstract service of which this application is a part. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

identifier

identifies the abstract service. The identifier must be a valid svc.*n*.service_id (0x20000-0xFFFFFFF for MSO) where that service is also described in the signaling.

app.n.visibility=visibility

controls whether the application is visible to the viewer. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

visibility

specifies if the application is visible to the viewer. Possible values for *visibility* are:

INVISIBLE

specifies the application is not visible.

VISIBLE

specifies the application is visible.

VISIBLE-TO-APPS-ONLY

specifies the applications can be used only by other applications.

Optional fields

Use the optional application fields to configure your application options.
Optional application fields are:

`app.n.api.index`

specifies the desired registered API number. Where:

n specifies a unique `app.n` number, starting with 0.
The initial range of values is from 0 to 31.

index specifies an increasing integer number. Assign each API a unique number. Possible values range from 0 to 15.

`app.n.application_name=name`

assigns the engineering name of the application. Where:

n specifies a unique `app.n` number, starting with 0.
The initial range of values is from 0 to 31.

name specifies the name associated with the application.

`app.n.application_name.argumentNumber=language ,name`

assigns the language- specific name expressed as *lang ,name*.
Where:

n specifies a unique `app.n` number, starting with 0.
The initial range of values is from 0 to 31.

argumentNumber

tracks the number of arguments in an increasing integer value.

language

specifies the language associated with the application.

name specifies the name associated with the application.

`app.n.args.argumentNumber=argument=setting`

Use the argument fields to initialize application parameters.
Arguments are passed through the properties file. Where:

n specifies an increasing integer number. Assign each application a unique `app.n` number, starting with 0. You can run as many applications within the limited space of the set-top box. The applications are run in successive order starting with `app.0` until all the assigned applications are run. By default, up to 32 unique applications may be defined.

argumentNumber

specifies an increasing integer number. Assign each argument a unique `args.n` number, starting with 0. The arguments are run in successive order starting with `args.0` until all the arguments are run. By default, up to eight unique arguments may be defined.

argument

specifies the argument. The argument is defined by each of the application separately.

setting specifies the value of the argument.

app.n.classpath_extension=path

specifies classpath extension directories. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

path specifies a semi-colon separated list of classpath extension directories.

app.n.icon_flags=path

specifies the icon flags as appropriate for AppIcon. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

path specifies the Unix-style absolute path to the icon flags.

app.n.icon_locator=path

specifies the relative path to the base_directory for icons. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

path specifies the Unix-style absolute path to the icon locator.

app.n.launch_order=order

specifies the order of applications with the same application identification and priority. Only the application entry with the highest launch order is entered in the application database. launch_order may be used to signal and store a new version of an application prior to a change of launch_order in a subsequent revision of the XAIT. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

order specifies a unique launching order. *order* is an 8-bit value.

app.n.service_bound=switch

determines whether the application is service bound or can continue to live with the next service. An application is always part of a service, but if not set, service is ignored. If assigned to TRUE, the application dies when a selected away from that service. If assigned to FALSE, the application lives until it is determined that the application can live in the next service. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

switch specifies a Boolean to indicate if the application is service bound. Possible values for *switch* are:

TRUE enables the option. TRUE is the default.

FALSE disables the option.

`app.n.storage_priority=priority`

specifies the application storage priority. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

priority

specifies the storage priority. Possible values range from 1 to 200.

`app.n.transport=number`

specifies the transport protocol indices. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

number

specifies a comma-separated list of transport protocols defined in `transport.n`.

`app.n.version=number`

specifies the version of the application. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

number

specifies the application version number.

Services

Use the service fields to configure the abstract service specifications for XAIT signaling. Abstract service specifications are valid in `xait`. Service fields have the following format:

`svc.n.field=value`

defines the abstract service. Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

field specifies the field for the abstract service.

value specifies the value associated with the specified *field*.

Required fields

Use the required services fields to define how and when the service is associated with the application. The required service fields are:

`svc.n.name=name`

determines if the service is automatically selected. Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

name specifies a name for the service.

`svc.n.service_id=value`

specifies an identifier for the service. Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

value specifies the type of application. Possible values for *value* is:

0x020000-0xFFFFFFF

specifies an XAIT application. This is required.

Optional fields

Use the optional service fields to configure your service options. The optional service field is:

`svc.n.auto_select=switch`

determines if the service is automatically selected. Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

switch specifies if the service is automatically selected. Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default.

Example file

The following is an example xait.properties file:

```
svc.0.service_id=0x2CAFE
svc.0.auto_select=true
svc.0.name=Registered Api test

app.0.application_identifier=0x000000016220
app.0.application_control_code=PRESENT
app.0.visibility=VISIBLE
app.0.priority=220
app.0.version=0
app.0.service=0x2CAFE
app.0.application_name=Api Test
app.0.base_directory=/sys cwd/qa/regapi/server
app.0.initial_class_name=com.vidiom.xlet.apiserver.ApiTestXlet
#app.0.args.0=onlysuccess

app.1.application_identifier=0x000000016221
app.1.application_control_code=PRESENT
app.1.visibility=INVISIBLE
app.1.priority=220
app.1.version=0
app.1.service=0x2CAFE
app.1.application_name=Test #1
app.1.base_directory=/sys cwd/qa/regapi/client
app.1.initial_class_name=com.vidiom.xlet.apiclient.ApiTestXlet
app.1.api.0=com.vidiom.api.apitest.TestApi
```

Configuring the hostapp.properties file

The host application form of signaling is used to describe resident host device applications.

The implementation reads a

hostapp.properties file at

boot-up and populates the services database based on its contents. The OCAP stack also uses the hostapp.properties file to keep track of your project's runtime settings. By default, this file contains the configuration settings for a single project. If you want to run multiple applications at once, you can define settings for as many applications as you can run within the limited space of the set-top box. If you need to run one or more other applications while your project runs, or if your application requires run-time arguments, you need to modify your project's hostapp.properties file. The hostapp.properties fields used for signaling are:

- ◆ *Version*
- ◆ *Transport protocols*
- ◆ *Applications*
- ◆ *Services*



The hostapp.properties file is located along the system class-path.

-
- ◆ **NOTE:** If you have more than one application in your hostapp.properties file, make sure each application has a unique app.n number and a unique application identifier.
-

Version

Use the version field to identify the application signaling version for host application signaling. If no definition is provided, then the implicit entry version=0 is assumed. The version field uses the following format:

version=n determines the version of the application signaling. Where:

n specifies the version level. version is a 5-bit field able to hold 0-31 decimal values.

-
- ◆ **NOTE:** Signaling updates are acknowledged only with a version change.
-

Transport protocols

Use the transport protocol fields to configure transport protocols for host application signaling. By default, if no transport protocols are defined, an implicit local protocol is defined as `transport.255=local`.

- ◆ **NOTE:** The local transport always has a label of 255, regardless of the index used in the definition. For example, the `app.i.transport` must reference 255 to get the local transport.

The transport fields have the following format:

`transport.n=protocol`

determines the transport protocol. Where:

n specifies an increasing integer number. Assign each transport a unique `transport.n` number, starting with 0. Possible values range from 0 to 31. To redefine the range of values, use `maxtp`.

`protocol`

specifies the type of protocol. Possible values for `protocol` are:

`oc` specifies the object carousel protocol.

`ip` specifies Internet protocol.

`local` specifies a local protocol.

`transport.n.key=value`

defines the value of the transport-stream identifier. Where:

n specifies an increasing integer number. Assign each transport a unique `transport.n` number, starting with 0. Possible values range from 0 to 31. To redefine the range of values, use `maxtp`.

`key`

specifies the variables for that transport stream protocol definition. Possible values for `key` are:

`remote=switch`

determines whether the protocol is remote. This is a required field when the type is `oc` or `ip`. Possible values for `switch` are:

`TRUE` enables the option.

`FALSE` disables the option.

`service=identifier`

specifies the abstract service identifier, but only if `remote=TRUE`. This is a required field when the type is `oc` or `ip`.

component=n

identifies the principal service component that delivers the application. This is a required field when the protocol is oc. component is the 8-bit component tag specifying the elementary stream that carries the DS1 of the object carousel.

alignment=switch

determine whether the alignment indicator is enabled. This is a required field when the type is ip. Possible values for switch are:

TRUE enables the option.

FALSE disables the option.

url.n specifies an increasing integer number.

Assign each URL a unique url.n number, starting with 0. Possible values range from 0 to 31. To redefine the number of URLs, use maxurl.

value specifies the value associated with the specified key.

Applications

Use the following format to configure the application fields:

app.n.field=value

specifies the application, the application's unique information, and the value associated with the information. Where:

n specifies an increasing integer number. Assign each application a unique app.n number, starting with 0. The initial range of values is from 0 to 31. You can run up to that many applications within the limited space of the set-top box. To redefine the range of values, use maxapps. The applications are run in successive order starting with app.0 until all the assigned applications are run. Assign each transport a unique transport.n number, starting with 0.

field specifies information about your set-top box applications.

value specifies the value associated with the specified field.

Application identifiers

Application identifier ranges are defined for AIT-signaled applications, for XAIT-signaled applications, and for manufacturer-defined host applications. For the hostapp.properties file, which signals host applications, valid application identifier ranges are:

- 0x0000-0x3fff
specifies unsigned applications.
- 0x4000-0x5fff
specifies signed applications.
- 0x6000-0x6fff
specifies dual-signed monitor applications.
- 0x7000-0x7fff
specifies dual-signed “super-host” (all-permission) applications.

Required fields

Use the required application fields to define how and when an application gets signaled. The required application fields are:

`app.n.application_control_code=mode`

determines how to control the application. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

mode specifies how to control the application. Possible values for *mode* are:

PRESENT

starts the application control now.

AUTOSTART

automatically starts the application control when the set-top box is booted.

KILL stops the application control.

REMOTE designates the remote control for application control.

`app.n.application_identifier=identifier`

uniquely identifies the application and its affiliated organization. The identifier value is used to keep track of running applications. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

identifier

uniquely identifies the application and its affiliated organization. The `application_identifier` is comprised of a 32-bit unique organization identifier (OID) and a 16-bit application identifier (AID). For example, 0x000000017001: OID=0x00000001, AID=0x7001. The application identifier determines the base level of permissions granted the application. Possible values for *identifier* are:

0x0000-0x3fff

specifies unsigned permissions.

0x4000-0x5fff

specifies signed permissions (plus additional permissions out of the application's PRF).

0x6000-0x6fff

specifies Monitor Application permissions (plus additional permissions out of the application's PRF).

0x7000-0x7fff

specifies all permissions.

`app.n.base_directory=path`

specifies the path to the application. Where:

n specifies a unique `app.n` number, starting with 0. The initial range of values is from 0 to 31.

path specifies the Unix-style absolute path to the application.

-
- ◆ **NOTE:** The `base_directory` path in the properties signaling can be absolute or relative. If relative, the path is considered to be relative to the defined system default directory. This is a port-dependent concept. On Win32 it corresponds to the current working directory, on other platforms it corresponds to the `FS.DEFSYSDIR` system property. This applies only if `application.0.transport` is undefined or defined as `application.0.transport=255` and `transport.255=local` is defined.

`app.n.initial_class_name=name`

defines the initial class name of the application implementation. Where:

n specifies a unique `app.n` number, starting with 0. The initial range of values is from 0 to 31.

name specifies the initial class name (`com.yourcompany.apps.YourXlet`) of the application implementation.

app.n.priority=priority

assigns resources to the set-top box. Higher priority applications get a larger share of available resources, such as CPU time and memory. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

priority specifies the resource priority of the application. Possible values for *priority* are:

255 specifies the highest priority. 255 is used by the Monitor Application.

100 to 254 specifies unbound applications.

1 to 200 specifies bound applications.

app.n.service=identifier

identifies the abstract service. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

identifier

identifies the abstract service. The identifier must be in the range 0x010000-0x01FFFF, which specifies a host application service defined by the manufacturer and signaled by hostapp.properties.

app.n.visibility=visibility

controls whether the application is visible to the viewer. Where:

n specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.

visibility

specifies if the application is visible to the viewer. Possible values for *visibility* are:

INVISIBLE

specifies the application is not visible.

VISIBLE

specifies the application is visible.

VISIBLE-TO-APPS-ONLY

specifies the applications can be used only by other applications.

Optional fields

Use the optional application fields to configure your application options. Optional application fields are:

app.n.application_name=name

assigns the engineering name of the application. Where:

n specifies a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31.

name specifies the name associated with the application.

app.n.application_name.argumentNumber=language ,name

assigns the language- specific name expressed as *lang ,name*. Where:

n specifies a unique *app.n* number, starting with 0. The initial range of values is from 0 to 31.

argumentNumber

tracks the number of arguments in an increasing integer value.

language

specifies the language associated with the application.

name specifies the name associated with the application.

app.n.args.argumentNumber=argument=setting

Use the argument fields to initialize application parameters. Arguments are passed through the properties file. Where:

n specifies an increasing integer number. Assign each application a unique *app.n* number, starting with 0. By default, up to 32 unique applications may be defined. You can run up to this many applications within the limited space of the set-top box. The applications are run in successive order starting with *app.0* until all the assigned applications are run.

argumentNumber

specifies an increasing integer number. Assign each argument a unique *args.n* number, starting with 0. The arguments are run in successive order starting with *args.0* until all the arguments are run. By default, up to eight unique arguments may be defined.

argument

specifies the argument The argument is defined by each of the application separately.

setting specifies the value of the argument.

`app.n.classpath_extension=path`
 specifies classpath extension directories. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- path* specifies a semi-colon separated list of classpath extension directories.

`app.n.icon_flags=path`
 specifies the icon flags as appropriate for AppIcon. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- path* specifies the Unix-style absolute path to the icon flags.

`app.n.icon_locator=path`
 specifies the relative path to the base_directory for icons. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- path* specifies the Unix-style absolute path to the icon locator.

`app.n.launch_order=order`
 specifies the order of applications with the same application identification and priority. Only the application entry with the highest launch order is entered in the application database. `launch_order` may be used to signal and store a new version of an application prior to a change of `launch_order` in a subsequent revision of the XAIT. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- order* specifies a unique launching order. *order* is an 8-bit value.

`app.n.service_bound=switch`
 specifies whether the application is service bound or can continue to live with the next service. An application is always part of a service, but if not set, service is ignored. If assigned TRUE, the application dies when a viewer selects away from that service. If assigned FALSE, the application lives until it is determined that the application can live in the next service. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- switch* specifies a Boolean to indicate if the application is service bound. Possible values for *switch* are:
 - TRUE enables the option. TRUE is the default.
 - FALSE disables the option.

`app.n.storage_priority=priority`
 specifies the application storage priority. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- priority* specifies the storage priority. Possible values range from 1 to 200.

`app.n.transport=number`
 specifies the transport protocol indices. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- number* specifies a comma-separated list of transport protocols defined in `transport.n`.

`app.n.version=number`
 specifies the version of the application. Where:

- n* specifies a unique app.*n* number, starting with 0. The initial range of values is from 0 to 31.
- number* specifies the application version number.

Services

Use the service fields to configure abstract service specifications for host application signaling. The OCAP specification designates a range of 24-bit identifiers for abstract services to differentiate them from 16-bit broadcast service identifiers. Abstract services can be defined by the host device manufacturer or by the MSO. The abstract service identifier range is subdivided to reflect the two means by which abstract services can be created:

`0x010000-0x01FFFF`
 specifies a host application service defined by the manufacturer and signaled by hostapp.properties.

`0x020000-0xFFFFFFF`
 specifies an application service defined by the MSO and signaled by XAIT.

In the hostapp.properties file, the specification of an abstract service must specify a manufacturer-defined abstract service; that is, the service identifier must be in the 0x010000-0x01FFFF range.

Service fields have the following format:

`svc.n.field=value`

defines the abstract service. Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

field specifies the field for the abstract service.

value specifies the value associated with the specified field.

Required fields

Use the required services fields to define how and when the service is associated with the application. The required service fields are:

`svc.n.name=name`

determines if the service is automatically selected. Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

name specifies a name for the service.

`svc.n.service=value`

determines the identifier for the service. Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

value specifies the type of application. Possible values for *value* are:

0x010000-0x01FFFF

specifies a host application

(manufacturer-defined) abstract service.

Optional fields

Use the optional service field to configure your service option. The optional service field is:

`svc.n.auto_select=switch`

specifies whether the service is automatically selected.

Where:

n specifies an increasing integer number. Assign each service a unique `svc.n` number, starting with 0. Possible values range from 0 to 31.

switch specifies if the service is automatically selected.

Possible values for *switch* are:

TRUE enables the option.

FALSE disables the option. FALSE is the default.

Example file

The following is an example hostapp.properties file:

```
app.0.application_identifier=0x000000017000
app.0.application_control_code=AUTOSTART
app.0.visibility=VISIBLE
app.0.priority=220
app.0.version=0x0
app.0.application_name=Launcher
app.0.base_directory=/sys cwd/apps/launcher
app.0.initial_class_name=com.vidiom.apps.launcher.AppLauncher$Xlet
app.0.args.0=showOnStart

app.1.application_identifier=0x000000010002
app.1.application_control_code=PRESENT
app.1.visibility=VISIBLE
app.1.priority=220
app.1.application_name=HSampler
app.1.base_directory=/sys cwd/apps/hsampler
app.1.initial_class_name=com.vidiom.apps.hsampler.HSampler2$Xlet

app.2.application_identifier=0x000000016002
app.2.application_control_code=PRESENT
app.2.visibility=VISIBLE
app.2.priority=220
app.2.application_name=watchTV
app.2.base_directory=/sys cwd/apps/watchtv
app.2.initial_class_name=com.vidiom.xlet.watchtv.WatchTVXlet
```

Configuring the Monitor Application

The Monitor Application is used by the MSO to exert policy over the operation of its OCAP network. It does this by having special privileges that permit it to control other applications. The Monitor Application allows the MSO to decide what applications can be on the network, the level of security the application has on the network, the resources to which the application has access, and the control of how those resources are used.

The Monitor Application is the first application started during the boot sequence and is associated with its own abstract service. This ensures that the initial Monitor Application has the opportunity to start up before other unbound applications are launched. The initial Monitor Application is normally signaled by the XAIT, requiring the OCAP stack to wait for an XAIT to be received before continuing the boot process. The OCAP stack may also be configured to support a fully resident host application as the initial Monitor Application. A Monitor Application must satisfy the following requirements:

- ◆ must be signaled in XAIT
- ◆ must have priority 255
- ◆ must be AUTOSTART
- ◆ must be signaled as part of an abstract service with `auto_select=true`
- ◆ must be the only application with AUTOSTART and `priority=255` in the abstract service

Signaling the Monitor Application

It may be desirable to use a resident Monitor Application but still signal a Monitor Application in the XAIT. For example, some set-top boxes may work best with a resident Monitor Application, while others work best with a network-signaled Monitor Application. For this reason you may want to support both a resident monitor and a network Monitor Application at the same time.

You can implement this support in one of two ways:

- ◆ using the same AppID for both Monitor Applications
- ◆ creating a smart network Monitor Application

Using the same AppID

OCAP allows only one instance of an application with a given AppID to be executed at one time. By using the same AppID for the resident and network Monitor Application the following will occur:

- ◆ The resident Monitor Application launches as part of system start up.
- ◆ The network Monitor Application's service is automatically selected, but the application will not launch if another application (the resident Monitor Application) with the same AppID is already running.

To prevent other applications from being launched unintentionally, the Monitor Application should be the only AUTOSTART application in the abstract service carrying the Monitor Application. If another application has AUTOSTART set, it is possible for the other application to be launched as there are no precedence rules for deciding which version, hostapp or XAIT application, is launched.

**Smart network
Monitor
Application**

You could have the network Monitor Application recognize if a resident Monitor Application is running. If it did recognize one, then the network Monitor Application could exit immediately. The actual implementation of this method is beyond the scope of this document.

**hostapp.properties
file**

Use the hostapp.properties file to configure a resident Monitor Application. The implementation reads a hostapp.properties file at boot-up and populates the services database based on its contents.

The hostapp.properties file is located in the %OCAPROOT%\bin%\%OCAPTC%\ directory. For example, for the SA-8300HD port, this file is located in the %OCAPROOT%\bin\Vidiom\SA\8300HD\debug\apps\ directory.

Use the resident Monitor Application field to configure a resident Monitor Application or associated support. The resident Monitor Application field have the following format:

`com.vidiom.ocap.monapp.resident=switch`

determines if the Monitor Application is resident. Where:

`switch` determines whether the Monitor Application is enabled. Possible values for `switch` are:

`false` disables the option. `false` is the default setting.

`true` enables the option.

For example:

`com.vidiom.ocap.monapp.resident=false`

- ◆ **NOTE:** To support a resident Monitor Application, configure the XAIT to be signaled by the hostapp.properties at boot time.

mpeenv.ini file

The Monitor Application can be signaled through the host application form of signaling used to describe resident host device applications. The implementation reads a `hostapp.properties` file at boot-up and populates the services database based on its contents. The `hostapp.properties` file includes the same information found in the `xait.properties` file.

The `mpeenv.ini` file is located in the `%OCAPROOT%\bin\%OCAPTC%\` directory. For example, for the SA-8300HD port, this file is located in the `%OCAPROOT%\bin\Vidiom\SA\8300HD\debug\mpeenv.ini` directory.

signaling support

Use the AIT and XAIT fields to configure AIT or XAIT support. XAIT fields have the following format:

`OCAP.ait.ignore=switch`

determines if AIT is ignored or acknowledged. Where:

`switch` specifies whether AIT is ignored or acknowledged.

Possible values for `switch` are:

`false` ignores the option. `false` is the default.

`true` acknowledges the option.

For example:

`OCAP.ait.ignore=false`

`OCAP.xait.ignore=switch`

determines if XAIT is ignored or acknowledged. Where:

`switch` specifies whether XAIT is ignored or acknowledged.

Possible values for `switch` are:

`false` ignores the option. `false` is the default.

`true` acknowledges the option.

For example:

`OCAP.xait.ignore=false`

4 Build Environment

Overview

The build environment for the OpenCable Application Platform (OCAP) stack requires a variety of software tools in predictable directories on the system. This chapter provides an overview of these tools and their required or preferred location in the directory structure. It also discusses the files used during the build and how to execute the build process. This chapter covers the following topics:

- ◆ understanding the directory structure
- ◆ understanding the software tools
- ◆ understanding build component files
- ◆ generating the build
- ◆ flashing the host device

The procedures described in this chapter are based on Vidiom's port of the OCAP stack to the Scientific-Atlanta (SA) SA-8300HD set-top box.

Before reading this chapter, you should have completed:

- ◆ installing the porting kit as outlined in chapter 2
- ◆ configuring the build environment as discussed in chapter 2

After reading this chapter, you should be:

- ◆ familiar with the directory structure of the porting kit
- ◆ familiar with the software tools used to build various parts of the OCAP stack
- ◆ able to identify and modify the files needed to complete a build
- ◆ able to generate the build
- ◆ able to flash the host device

Understanding the directory structure

The porting kit files are contained within a parent directory that you selected during the installation of the porting kit. These files are organized into several top-level directories under the parent directory. This section discusses the directory structure and the types of files found in each top-level directory (by default C:\OCAP1-PK).

Understanding the purpose of each of these directories makes it easier to locate files within the porting kit.

%OCAPROOT% (parent) directory

%OCAPROOT% is an environment variable that points to the directory where the porting kit files were installed. The default directory is C:\OCAP1-PK, but because the location of this directory is user-dependent, the build environment calls the %OCAPROOT% to locate the files it needs. %OCAPROOT% is equal to the drive letter plus all subdirectories between the drive root and the OCAP parent directory. For example, on a Windows system where the porting kit is installed to a directory named Vidiom in the Program Files folder, the value of %OCAPROOT% might be C:\Program Files\Vidiom\OCAP1-PK.

Top-level directories

This section lists the top-level directories created during the installation of the porting kit and the contents of each. Top-level directories are installed in the %OCAPROOT%\, are as follows:

- ◆ *apps (applications)*
- ◆ *assets*
- ◆ *bin (binaries)*
- ◆ *docs*
- ◆ *extensions*
- ◆ *java*
- ◆ *jni*
- ◆ *jvm*
- ◆ *mpe*
- ◆ *other*
- ◆ *target*
- ◆ *tools*

**apps
(applications)**

You can use the sample OCAP applications for testing or as code examples. The sample applications are Java files and are located in the apps directory. Currently, the sample application directories include:

config includes the HConfig application, which configures the setting for high definition in the set-top box. For example, selecting video output resolution.



For more information about HConfig, refer to the %OCAPROOT%\apps\config directory.

launcher includes the AppLauncher application, which displays a simple menu allowing you to launch other applications. The AppLauncher queries the applications database (`hostapps.properties` file) to determine the available applications and access their attributes, including the name.

stupid includes a very simple Xlet used as a placeholder for testing bound and unbound applications.

assets

The assets directory is used to store the resource files for the OCAP stack. The assets directory should be used to store art (for example, .bmp or .ico files), fonts, mpeg files used by the OCAP stack. Currently, the following asset directory is included:

fonts includes the Bitstream subdirectory. Bitstream contains the Tiresias font used for English text.

bin (binaries)

The final binaries for the target configurations are created in the bin directory. Currently, the following directory is included:

Vidiom specifies the different operating systems and the targets associated with the operating systems. Currently, the following operating system is included:

SA includes the binaries for Scientific-Atlanta targets (for example SA-8300HD).

docs

Documentation for the OCAP project is placed in the docs directory. Documentation includes the Java documentation, license agreements, release notes, software design specification, and this porting guide.

extensions

The extensions directory stores any third-party extensions to the OCAP stack that require Java Native Interface (JNI) support. Currently, the following extensions is supplied:



For more information on OCAP JNI extensions, refer to the OCAP extensions section.

| | | |
|--------------|---------------------|---|
| | dvrupgrade | provides an application that manages legacy recording by determining if legacy recordings exist on the set-top box, converting the legacy recording to OCAP recordings, and managing the list of selectable recordings. The subdirectories build separate Java and JNI libraries: |
| | java | provides the Java package using native methods to access the JNI library. |
| | test | provides a test to insure the dvrupgrade functionality is working properly. |
| | jni3pExample | provides an example of how a third-party can create an OCAP extension that requires native JNI support. The subdirectories build separate Java and JNI libraries: |
| | dll | provides an example JNI library to implement the example native methods. |
| | java | provides an example Java package using native methods to access the JNI library. |
| java | | The Java portions of the OCAP stack are maintained in the java directory. The majority of the code is placed in the base subdirectory. Other subdirectories may be used to provide implementations of specific classes for different target configurations or the Java Virtual Machine (JVM). |
| jni | | All JNI header files and implementations are placed in the jni directory. The organization of this directory allows for JNI implementations corresponding to the base and variant packages in the java directory. |
| jvm | | The jvm directory is not created during the initial installation of the OCAP stack. You create this directory when installing the virtual machine. Its purpose is to hold the various implementations of the JVM and core Java libraries. The directory structure here should indicate the vendor and product names but no rigid naming convention is required. |
| mpe | | Multimedia Platform Environment (MPE) and Multimedia Platform Environment Operating System (MPEOS) source files are placed in the mpe directory. The MPE layer is the native interface to which the JVM and Java libraries are written. The MPE interface is defined by the header files in mpe\include directory and most of the work is performed by managers in mpe\mgr . |
| | | The MPEOS layer is the layer that must be ported for each target platform. The MPEOS interface is defined by the header files in mpe\os\include and a subdirectory for each target platform is present in mpe\os . |
| other | | Third-party libraries used to implement the OCAP stack are placed in the other directory. Currently, the following third-party libraries are included in the porting kit: |

| | | |
|------------|--|---|
| ares-1.1.1 | includes the asynchronous resolver library (ares). Ares is intended for applications that need to perform Domain Name Server (DNS) queries without blocking, or need to perform multiple DNS queries in parallel. The primary examples of such applications are servers which communicate with multiple clients and programs with graphical user interfaces. |  For more information about the asynchronous resolver library, refer to the README file located in the %OCAPROOT%\other\ares-1.1.1 directory. |
| DirectFB | includes DirectFB, which provides developers with hardware graphics acceleration, input device handling and abstraction, integrated windowing system with support for translucent windows, and multiple display layers. DirectFB is a complete hardware abstraction layer with software fallbacks for every graphics operation that is not supported by the underlying hardware. |  For more information about DirectFB, refer to the README_VIDIO.M.txt file located in the %OCAPROOT%\other\DirectFB directory. |
| FreeType2 | includes the FreeType2 software font engine, which is designed to be small, efficient, easy to customize, and portable while capable of producing high-quality output (glyph images). It can be used in graphics libraries, display servers, font conversion tools, text image generation tools, and many other products. |  For more information about FreeType2, refer to the %OCAPROOT%\other\FreeType2\docs directory. |
| Zlib-1.2.1 | provides the MPE data/object carousel implementation a general purpose data compression library. The Zlib compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data. |  For more information about Zlib-1.2.1, refer to ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html . |

| | |
|---------------|---|
| target | Configuration files for the target platforms are kept in the child directories in the target directory tree. The first level subdirectories under target are named for the porting company (for example, \Vidom\). There may be one or more levels of subdirectories under this host name that indicate different products or versions of products available (for example, \Vidom\SA\8300HD). Finally, under each product or version folder, there are folders for types of builds available (for example, \SA\8300HD\debug or \SA\8300HD\release). |
| tools | <p>Software tools included in the porting kit are placed in the tools directory. Currently, two subdirectories (generic and Win32) are under the tools directory that group tools by the host systems they are used on.</p> <p>Tools that are independent of the host platform are placed in tools\generic directory. Currently, the following generic tool directories are included in the porting kit:</p> <ul style="list-style-type: none"> apache-ant-1.6.0 includes the Ant tool, which automates the build process. bin includes scripts used by build process. Some of the files used in the build process are not tied to any one operating system and are not a part of any other package. GroboUtils-1 includes the GroboUtils open-source software package that can be used in writing Java unit tests. jakarta-log4j-me includes the LOG4J tool, which inserts log statements into your code for debugging. junit3.8.1 includes the JUnit tool, which is an open source Java testing framework used to write and run repeatable tests. NanoXML-2.2.3 includes the NanoXML tool, which is a small non-validating parser for Java. pbp1.0 includes the Java Archive (JAR) files used to build the OCAP stack. The OCAP stack is built against these files instead of the ones shipped with the Esmertec Virtual Machine (EVM) to eliminate the build dependency on the EVM. proguard2.1 includes the ProGuard tool, which detects and removes unused classes, fields, methods, and attributes. It can then optimize bytecode and remove unused instructions. Finally, it can rename the remaining classes, fields, and methods using short meaningless names. The resulting jars are smaller and harder to reverse-engineer. |

`xerces-2_4_0` includes the Xerces2 tool, which is a high performance, fully compliant, Extensible Markup Language (XML) parser in the Apache Xerces family. This version of Xerces introduces the Xerces Native Interface, a complete framework for building parser components and configurations that is extremely modular and easy to program. Xerces2 is used for JUnit testing and is included in the `support-test.jar`.

Windows hosted tools are placed in `tools\Win32`. Currently, the following Windows tools directories are included in the porting kit:

`bin` includes any generic Windows executables needed for the OCAP project. This includes the `omake.exe` utility, which is used to control the OCAP build process.

`cygwin` includes the Cygwin tool and a port of GNU Make, which is used to conduct makefile-based builds of some of the native components of the OCAP stack.

`j2sdk1.4.2_13` includes the Java 2 v1.4.2 Software Development Kit (SDK), which is a development environment for building applications, applets, and components using the Java programming language.

`jdk1.3.1_07` includes the Java 2 v1.3.1 SDK, which is a development environment for building applications, applets, and components using the Java programming language. This version of the SDK is used primarily for JNI includes and the Java compiler (`javac`).

target operating system (for example, SA)
includes the SDKs for the target host device.

Understanding the software tools

The build environment for the OCAP stack uses a variety of software tools to build various parts of the OCAP stack. Most of the tools required by the porting kit are included on the disk(s), for example, Cygwin. Some proprietary tools must be licensed from their manufacturers. In addition, some SDKs provided by platform retailers may require additional tools, which may or may not be included on their disks.

The tools used can be divided into two functional categories:

- ◆ those that drive or control the build process itself, or build control tools
- ◆ those that perform the actual code-specific build operations, or build construction tools

You must install each tool at the location expected by the build control tools. This section provides descriptions and installation locations for each of the required tools.

Build control tools

Build control tools are used to drive the build process. The build control tools required for the OCAP stack are included in the porting kit. This section provides descriptions and installation requirements for each of the tools. The following build control tools are required:

- ◆ *Ant*
- ◆ *omake*

Ant

Ant is the build tool used to manage the overall build process. Ant is a Java-based make tool that uses XML files to describe and execute the build. The build file, which is usually named `build.xml`, defines the build through a set of targets and related target trees where various build-related tasks get executed. The targets and their trees can have additional build configuration files that further define sub-targets and methods for performing the builds for those sub-targets. This strategy easily supports a hierarchical software build process that matches the requirements for building a hierarchical software architecture such as the OCAP stack.

-
- ◆ **NOTE:** The required Ant distribution is installed during the porting kit installation in the `%OCAPROOT%\tools\generic\apache-ant-1.6.0` directory

omake

omake is a build environment configuration utility written for the OCAP stack development project. It ensures a consistent build process by setting temporary environment variables and handshaking with other tools. You start a build of all or a portion of the OCAP stack by executing omake on the command line.

When invoked, omake performs several environment-related tasks to clear the execution environment (for example, modifying the path statement or setting temporary environment variables) and then configures the environment based on the target configuration. This configuration process helps ensure that any conflicting versions of tools resident on the host system do not interfere with the build process for a particular target.

-
- ◆ **NOTE:** The required omake distribution is installed during the porting kit installation in the %OCAPROOT%\tools\Win32\bin directory. To use the omake utility, add the path of the omake executable to your path statement.
-

Build construction tools

Construction tools perform the actual code-specific build operations. In addition to these tools required for the OCAP stack, there may be additional and/or proprietary tools as part of the platform-specific SDK provided by platform retailers. The following build construction tools are required:

- ◆ *ActivePerl*
- ◆ *Cygwin*
- ◆ *Target-specific SDKs*

ActivePerl

ActivePerl is an open version of Perl for the Windows operating system. ActivePerl is developed and administered by ActiveState. Perl scripts are used in various aspects of the Esmertec JVM build process, but also appear in additional, target-specific build steps for other components of the OCAP stack.

-
- ◆ **NOTE:** To acquire and download ActivePerl, go to <http://www.activestate.com>. Install ActivePerl into the C:\Perl directory on the host development system.
-

Cygwin

The Cygwin tool set, including a port of GNU Make, is used to conduct makefile-based builds of some of the native components of the OCAP stack. In addition, some other utilities are used for build related operations such as deleting files and directories.

-
- ◆ **NOTE:** The required Cygwin distribution is installed during the porting kit installation in the %OCAPROOT%\tools\Win32\cygwin directory. When using the OCAP version of the Cygwin tools, be sure you do not have another version (C:\Cygwin) currently running. Doing so may cause incompatibilities with the OCAP version of these tools.
-

Target-specific SDKs

Target-specific SDKs are available from host device manufacturers for each system. Typically, an SDK includes all of the definition files, build files, and compiler-related tools to develop an application for the target platform.

A significant part of porting the OCAP stack to a target SDK is to identify and extrapolate the build elements required for the build process to work. This includes noting information such as the C include header files location, where the compiler tools are located, what compiler flags and options to use, and how to package an executable for execution on the target platform. The OCAP stack build environment has been designed to allow for definition and inclusion of all these elements of a third-party SDK build environment.

-
- ◆ **NOTE:** The SDK manufacturer will supply the procedures to install a third-party SDK. The OCAP stack provides a logical location and organization for these SDKs. The standard location for a third-party SDK is the `tools` directory (for example, `%OCAPROOT%\tools\Win32\host device operating system\SDK name and version`).
-

Understanding build files

The OCAP stack build environment uses Ant as its main build control tool. The build relies on a hierarchical series of XML files, make files, and make rules files, which in turn call SDK rule, property, and file-list files for the construction of OCAP stack components. Each component plays a specific role within the build process. The requirements of a target configuration's sub-components determine these roles.

While many build files are used in the OCAP stack build process, only a portion may need modification when performing a port. The most important build component files are:

- ◆ *build.xml files*
- ◆ *buildrules.properties files*
- ◆ *buildrules.mak file*
- ◆ *sdkrules.mak files*
- ◆ *buildenv.bat files*
- ◆ *deploy.properties files*
- ◆ *Makefile files*

build.xml files

Because the OCAP stack build environment is based on Ant, the primary files for the build process are the Ant-based *build.xml* files. Information to drive the build for a target configuration is specified in these files.

Each target configuration within the \target directory contains a *build.xml* file that will build the OCAP stack for that configuration. Ant allows a *build.xml* file to contain other base targets (for example, MPE, MPEOS, JVM, Java, etc.). You may specify these base targets by including them on the command line when starting the build.

Base targets are typically either an actual code component of the OCAP stack or an additional Ant *build.xml* file to be processed at a different level of the development tree. For example, a native C-language based code target usually specifies an executable for the build and a specific command for running the executable to generate the sub-target component (for example, a shell command to run a make with a target *makefile*). For additional sub-targets, the upper-level *build.xml* target specification defines the sub-target and an associated sub-target specific *build.xml* file within the development tree.

The base or master `build.xml` file for a platform target is always located within a target configuration subdirectory for the platform. All of the target configuration directories are located within `%OCAPROOT%\target\Vidiom\%OCAPTC%`. For example, a target platform specifying a debug and release target configuration would have the following target configuration directories:

- ◆ `%OCAPROOT%\target\Vidiom\target platform\target host device\debug`
- ◆ `%OCAPROOT%\target\Vidiom\target platform\target host device\release`

The *target platform* directory usually corresponds to a specific manufacturer, for example, Scientific Atlanta. The *target host device* usually corresponds to a specific set-top box or television, for example, 8300HD. This is usually the case because most set-top box platforms have distinct features and often specific SDKs, although it is possible a single SDK may target multiple set-top box platforms. But, typically, the set-top box platforms will have different features with different operating-system binaries that justify segregation into separate *target host device* directories.

buildrules.properties files

The `buildrules.properties` files, as the name implies, defines various build properties that are fed into the build process. These properties are additional parameters used by the build process for determining how different target components are built. The format of the parameter definitions resembles that of other standard Java property files.

The properties may constitute pre-build, build, or post-build operations and/or operational parameters. For example, the properties for how to build a debuggable version of a component may be entirely different than building a release version because it may involve different or additional steps to enable, include, or supply debug information. There may be different libraries, modules, or JAR files that have to be processed or copied for a debug versus a release version of a target component.

Exactly what properties need to be defined within the `buildrules.properties` file are specific to the requirements of the target component being built and to the particular configuration of that target.

The `buildrules.properties` file is located in the base subdirectory for a platform target configuration (for example, `%OCAPROOT%\target\Vidiom\%OCAPTC%\buildrules.properties`).

buildrules.mak file

The buildrules.mak file defines various parameters that are fed into all makefiles. The parameters are typically make rules for how to build various components. The format of the parameter definitions is simply the format used for makefile macro definitions within actual makefiles. Common rules included are:

- ◆ compiler command line options
- ◆ compiler macro definitions
- ◆ include path definitions
- ◆ common link-stage library references
- ◆ common link-stage command arguments

In actuality, the specific compilation rules are not directly specified within the buildrules.mak file itself. Rather, the buildrules.mak file acts as a conduit for the compilation rules that are provided by an SDK-specific build file. This file, normally called the sdkrules.mak file, is located within the associated SDK directory for the target configuration platform.

The buildrules.mak file is located in the base subdirectory for a platform target configuration (for example, %0CAPR0OT%\target\Vidiom\%0CAPTC%\buildrules.mak).

sdkrules.mak files

The sdkrules.mak files provide the specific compiler and executable packaging build step parameters for sub-targets involving SDK-specific build operations. These files typically specify the following:

- ◆ the actual compiler commands and execution paths
- ◆ the common compiler options
- ◆ common header and library file search paths
- ◆ link-stage parameters
- ◆ any special packaging commands and options that may be required to process the various binary components into a format suitable for loading or flashing into the target platform host device

The exact contents of these files are highly platform-SDK dependent. In general, they contain all of the SDK-related values that must be fed into the build process that are common to all of the target platforms that use a particular SDK. The sdkrules.mak file contents are usually directly derived from makefiles of the target SDK, and in fact, they often are original makefiles that are pared down to the essential parameters and then directly included into the target configuration buildrules.mak file via include statements.

The sdkrules.mak file is located in the same directory as the SDK for the target configuration (for example, tools\Win32\target platform\target SDK\vsdkrules.mak).

buildenv.bat files

The buildenv.bat main purpose is to call combuildenv.bat. The buildenv.bat file may also include any environment variable configuration commands specific to the type of build (debug or release).

combuilenv.bat files

The combuilenv.bat is a Windows batch execution file containing common environment variable configuration commands. The environment variable settings defined are typically values specific to a platform target configuration and specify various parameters of the build. Often included environment variables are SDK-specific variables, SDK-specific extensions to the PATH variable, and target-specific variables. This batch file is effectively executed by the omake utility to set up and configure the build execution environment before the build process.

deploy.properties files

The deploy.properties file serves only one purpose specific to the Esmertec JVM; it specifies the byte/bit ordering (“endianness”) of the target processor and must be in place when building the EVM.

Makefile files

There are a number of makefiles (named Makefile) throughout the OCAP stack development tree. These makefiles perform the standard role of a makefile, in that they serve as the mechanism for executing the native C-language based compilation build steps. The makefiles specify:

- ◆ the build target
- ◆ the target dependencies
- ◆ the build rules for compiling the target components

The bulk of the actual compilation build rules and parameters come from the previously mentioned buildrules.mak and sdkrules.mak files for the target configuration. But, often times, additional parameters are specified within the actual sub-target makefiles that extend the common parameters. For example, the base buildrules.mak and sdkrules.mak files specify the common header file include paths and common compile-time macros to define for the build, but the makefile for a specific sub-target may contain additional include-file paths or additional compile-time macro definitions relevant only to the sub-target. Most makefiles are generic and do not require modification for a specific target platform. Only target-specific makefiles (for example, those in %OCAPROOT\mpe\os\targetplatform) should be modified.

Generating the build

After completing the essential steps of a port, including the integration and configuration steps for the build environment, the OCAP stack should be ready for execution of the build process. The omake-based build process is very versatile and powerful. Not only can it be used to build all or any portion of the OCAP stack targets, it can also assist with verifying or debugging the configuration of the build environment. Additionally, you can execute the omake command-line utility using a command shell from any location within the OCAP1-PK directory structure.

Syntax

The command-line syntax for executing the omake utility is as follows:

```
omake [option...] build target [build target...]
```

Build options

The values defined for the build option are:

- batch executes a batch file (build.bat by default) to perform the build.
- debug displays debugging information as operations are performed. You can use the -debug option, which causes additional information to be echoed during the build, to analyze the build process for a target.
- emacs produces logging information without adornments.
- env displays environment information and returns to the command prompt. You can use the -env option to verify the build environment has been configured correctly. The -env option displays the entire environment variable state that is in place during the build.
- f *file* specifies the build *file* to use.
- find *file* searches parent directories until the build *file* is found.
- h displays the list of options.
- make spawns omake instead of Ant to perform the build.
- nice executes the build in nice (low priority) mode.
- shell spawns a shell and exits when it returns. The -shell option allows for execution of commands from a new command shell after the build environment has been configured. You can use the -shell option to verify that the build environment has been configured correctly.
- t lists all supported build targets.
- tc *config* specifies the target configuration (overrides %OCAPTC%) to build.
- u extends environment with user batch file %HOME%\omake.bat.

-v displays verbose information about operations being performed. You can use the -v option, which causes additional information to be echoed during the build, to analyze the build process for a target.

Build targets

If you do not specify a specific build target defined within the relevant build.xml file on the command line, the default build target specified within the build.xml file will be built. Other useful build targets specified by the base build.xml file are:

| | |
|------------|--|
| build | builds all the code and is currently the default value. |
| build.mpe | builds only the MPE portion of the OCAP stack, including the JNI functions that call MPE Application Programming Interfaces (API). |
| build.java | builds only the OCAP-specific Java portions of the OCAP stack. |
| build.jvm | builds only the JVM portion of the OCAP stack. |
| build.qa | builds the MPE targeted unit testing support. |
| clean | removes all temporary and intermediate files generated by the build process. |
| purge | removes all of the binaries generated by the build process. |

When the build has successfully completed, the ocap_image.rom file is created and is located in the target directory where the build was initiated (for example, %OCAPROOT%\target\Vidiom\%OCAPTC%).

Installing the OCAP stack on the host device

Installing the OCAP stack on the host device is done after portions of the MPEOS APIs have been ported and built. The following are the instructions for preparing and installing the host device to run the OCAP stack.

Before installing

Before installing the OCAP stack onto the host device, you need to:

1. Verify the host device is a development set-top box. The Ethernet and/or serial ports on this set-top box are necessary for loading the porting kit into memory.
2. Configure the serial port with the following settings: set the port speeds to 115,200 bps, 8 data bits, no parity, 1 stop bit, no flow control.
3. Build the `ocap_image.rom` image.



For information on creating the `ocap_image.rom`, refer to the Generating the build section.

Connecting the hardware

Connections between the host device, development system, and possibly the television are required. Connect the hardware as follows:

1. Connect the host device to a television by connecting the High Definition Television (HDTV) Y-component video out to the composite video input on the television. Audio connections do not need to be made.
2. Connect the Serial II port from the host device to an RS-232 port on your development system.
3. Connect the Ethernet port from the host device to your local IP network at your location.

Flashing the host device

Once a portion of the MPEOS APIs are ported and the OCAP stack is built, use `ocap_image.rom` to fully embed the OCAP stack in the host device. You can flash the host device using an Ethernet or a serial cable.



For detailed instructions (Ethernet or serial) on flashing the OCAP stack onto your host device, refer to the host device documentation.

Using an Ethernet cable is the fastest way to flash a host device. If the device does not have an IP address, the installation must be done through the serial port.

OCAP extensions

Some OCAP ports may implement non-standard extensions to OCAP for accessing specific native functionality in a set-top box (for example, front-panel lights, etc.). The OCAP extension are located in the %OCAPROOT%\extensions\ directory. There are currently two extensions:

- ◆ Jni3pExample
- ◆ dvrupgrade

Jni3pExample OCAP extension

Jni3pExample provides an example of how a third-party can implement its own OCAP extensions into the OCAP stack. The files are located in the %OCAPROOT%\extensions\Jni3pExample directory. The Jni3pExample directory includes the following subdirectories:

java provides an example Java package using native methods to access a JNI library.

- ◆ **NOTE:** Real classes should be in the appropriate namespace (for example, com.company.xxx).
-

dll provides an example JNI library to implement the example native methods.

Legacy DVR recording extension

This section provides descriptions for the legacy DVR recording extension implementation and `mpeenv.ini` configuration.

Understanding the implementation

The Legacy DVR Recording extension provides an API to allow a resident host application or Xlet with `MonitorAppPermission("handler.recording")` permissions to upgrade legacy (pre-OCAP) recordings to OCAP recordings. This extension is packaged as `com.vidiom.ocap.dvrupgrade`.

The Xlet must implement the `DVRUpgradeTranslator` interface, which currently consists of two public methods:

- ◆ `int translate (String recordingId, DVRUpgradeFileAccess fAccess)`
- ◆ `DVRUpgradeRecordingInfo getRecordingInfo()`

The Xlet obtains an instance of `DVRUpgradeManager` from `DVRUpgradeManager.getInstance()`. Access to `DVRUpgradeManager` is limited to resident host applications or Xlets that have been granted `MonitorAppPermission("handler.recording")` which provides global access to DVR recordings.

The Xlet then queries

`DVRUpgradeManager.isDefaultRecordingManagementEnabled()` to determine if the set-top box is in upgrade mode or default recording management mode.

If the set-top box is in the default recording management mode, then `DVRUpgradeManager` cannot be used to translate legacy DVR recordings. If the set-top box is in upgrade mode, `DVRUpgradeManager` can be used to translate legacy DVR recordings.

-
- ◆ **NOTE:** The Xlet invokes `DVRUpgradeManager.upgrade()` to initiate the upgrade, passing a reference to itself (recall that the Xlet implements the `DVRUpgradeTranslator` interface). The `DVRUpgradeManager` then invokes the `DVRUpgradeTranslator.translate()` interface method, iteratively, for each legacy recording.
-

`translate()` is provided with a `recordingId` that corresponds to the identifier returned to the OCAP stack by the underlying operating system and an object reference to an instance of `DVRUpgradeFileAccess`. This object is provided so Xlets can access the needed legacy DVR recording database files. Access to these files is read-only. The directories that are allowed to be accessed are specified in the `mpeenv.ini` file (which is described in the *mpeenv.ini Configuration* section).

`translate()` attempts to recover the information needed to create an entry in the OCAP DVR recording database. This information is encapsulated in the `DVRUpgradeRecordingInfo` class. `translate()` returns a value indicating how `DVRUpgradeManager` will handle that recording. Currently, possible values include:

`DVRUpgradeTranslator.TRANSLATE_YES`

specifies the `DVRUpgradeManager` adds this recording to the OCAP DVR recording database.

`DVRUpgradeTranslator.TRANSLATE_NO`

specifies `DVRUpgradeManager` does not add this legacy DVR recording to the catalog of OCAP DVR recordings. But, it should keep the recording.

`DVRUpgradeTranslator.TRANSLATE_DELETE`

specifies the `DVRUpgradeManager` deletes this legacy DVR recording.

If `translate()` returns `TRANSLATE_YES`, the `DVRUpgradeManager` invokes `getRecordingInfo()`. `getRecordingInfo()` returns an object reference to an instance of `DVRUpgradeRecordingInfo` containing the information associated with the recording that is to be added to the OCAP recording database.

The Xlet may provide multiple translators. A single Xlet can have multiple translators, one for each legacy system that it may encounter. The Xlet should invoke `DVRUpgradeManager.upgrade()` for each of its translators. The Xlet can query `DVRUpgradeManager.getNumRecordings()` to determine if there are any more recordings to be translated. If there are more recordings, then the Xlet can invoke `DVRUpgradeManager.upgrade()` for the next translator.

When the translations are complete, the Xlet can invoke `DVRUpgradeManager.setDefaultRecordingManagementEnabled(String logInfo)` to put the OCAP stack into the default recording management mode. The OCAP stack determines whether it is in the default recording management mode by testing for the presence of a flag file. The `logInfo` argument provides text that will be written to this flag file and used to save log information, the date of translation, and so on.

Each time the OCAP stack is initialized (through power-cycle, for example), the stack determines if there are legacy recordings. If there are no legacy recordings and the stack is not in the default recording management mode, the OCAP stack automatically transitions to default recording management mode.

mpeenv.ini Configuration

The following environment variables must be modified when using the com.vidiom.ocap.dvrupgrade extension:

-
- ◆ **NOTE:** There are no spaces before or after the equals sign.

VMOPT.0 The default virtual-machine classpath must be extended to include the jar file that holds the com.vidiom.ocap.dvrupgrade class files. The default location of this jar file is:

```
%OCAPROOT%\bin%\OCAPTC%\sys\dvrupgrade.jar
```

Simply append the corresponding pathname to the definition of VMOPT.0.

For example, for the ROMFS configuration:

```
VMOPT.0=-Djava.class.path=/romfs/usr;/  
romfs/apps;ocap.icb; /romfs/sys/ocap-rez.jar;  
/romfs/sys/support.jar;/romfs/sys/dvrupgrade.jar
```

OCAP .extensions

com.vidiom.ocap.dvrupgrade must be added to the list of OCAP.extensions.

For example:

```
OCAP.extensions=com.vidiom.debug;com.vidiom.ocap  
.dvrupgrade
```

com.vidiom.impl.manager.recording.

DefaultRecordingManagementEnabled

The default location of the default recording management flag file is %dvb.persistent.root%/DefRec. To specify a different filename, add an entry like the following to mpeenv.ini:

```
com.vidiom.impl.manager.recording.  
DefaultRecordingManagementEnabled=  
DifferentFilename
```

-
- ◆ **NOTE:** This entry is not required.

com.vidiom.ocap.dvrupgrade.DVRUpgradeFileAccess

If the application or Xlet needs to use the instance of DVRUpgradeFileAccess, which is passed to translate(), an appropriate list of accessible directories must be defined. The list consists of one or more semicolon-separated absolute pathnames.

For example:

```
com.vidiom.ocap.dvrupgrade.DVRUpgradeFileAccess=  
/AcmeProprietaryData;/RoadRunnerData
```

-
- ◆ **NOTE:** With this entry in `mpeenv.ini`, any resident host application or `MonitorAppPermission("handler.recording") Xlet` will be granted read-only access to the listed directories and their contents, including subdirectories. The purpose of this class is to provide disk file access for those translators that do not otherwise have access to the necessary disk files.
-

Creating third-party OCAP extensions

Some OCAP ports may implement non-standard extensions to OCAP for accessing specific native functionality in a set-top box (for example, front-panel lights, etc.).

Create the OCAP extension by following these steps:

1. Create a subdirectory where the source and build files reside (for example, %OCAPROOT%\extensions\OCAP extension name).
2. Using the example in the Jni3pExample subdirectory, create the Java and JNI source and build files for the new OCAP extension.
3. Once your environment is configured correctly, use the `omake clean` `purge` `build` command from the extension subdirectory to force a re-build of the OCAP extension, Java, and native JNI libraries.
4. Add your new extensions directory to the %OCAPROOT%\extensions\build.xml file so that it gets rebuilt whenever you perform a full target build (for example, a build from the %OCAPROOT%\target\Vidiom\%OCAPTC% directory).
5. Update the Java Virtual Machine (JVM) property `com.vidiom.ocap.extensions` to include the new Java package. This update exposes the new OCAP extension to OCAP applications. This property is located in the %OCAPROOT%\bin\Vidiom\%OCAPTC%\mpeenv.ini and looks like:

```
vmopt.22=-Dcom.vidiom.ocap.extensions=com.vidiom.debug,com.vidiom.jni3pExample
```



For detailed information on setting up your environment, refer to [Configuring the development system](#) section.

5 MPEOS Overview

Overview

The Multimedia Platform Environment Operating System (MPEOS) porting layer is a set of Application Programming Interfaces (APIs) providing mappings to operating-system specific APIs for functionality required by the Multimedia Platform Environment (MPE), Java Native Interface (JNI), and Java Virtual Machine (JVM) portions of the OpenCable Application Platform (OCAP) stack. In some cases where the functionality is very elementary (for example, acquiring the system time), the MPEOS APIs are direct mappings to the relevant operating system APIs. In other cases where the functional requirements of the upper, platform-independent portions of the OCAP stack are more than what is directly supported by the target operating system, the MPEOS APIs supply code to fill in the gaps in functionality.

Before reading this chapter, you should:

- ◆ have an understanding of the target operating system

After reading this chapter, you should be:

- ◆ familiar with the strategy for building the MPEOS layer
- ◆ familiar with the priority in which to port the different APIs
- ◆ familiar with the header files that require target-specific definitions

Strategy

The sources for the MPEOS porting layer APIs are organized within subdirectories of %OCAPROOT%\mpe\os. Each target operating system subdirectory contains all of the source code, header files, and build files specific to that port. For example, the Scientific Atlanta, PowerTV operating system port subdirectory is organized as follows:

`mpe\os\powertv`

contains the `mpeos_API name.c` source files and the `Makefile` for building the MPEOS library.

`mpe\os\powertv\include`

contains the `os_API name.h` header files associated with the `mpeos_API name.c` source files.

The strategy for building the MPEOS layer includes building the MPEOS APIs into an intermediate library to be linked against other software in later portions of the



For detailed information on testing the ported APIs, refer to the OCAP Testing Guide.

build process. The advantage to packaging the MPEOS API layer as a library is that, in addition to supporting the OCAP stack, the APIs can be used for direct unit testing during the very early stages of a port. The MPEOS unit test code uses the MPEOS API library to enable verification of the ported APIs.

The `Makefile` (`.mak`) located within an operating system subdirectory lists all of the source and header file dependencies for building the MPEOS library, including any target-specific header files used to resolve types and references used within the port. The target-specific header files are usually included into the actual build process via the `#include` statement with the appropriate `mpe\os\target operating system\include\mpeos_API name.h` header file. Additionally, the `Makefile` includes the basic rules for building the library (for example, the compiler include paths unique to constructing the MPEOS layer, etc.).

The library file is typically built and placed by the build process under the `%OCAPROOT%\gen` directory within a `lib` subdirectory of the target platform. For example, if the library target name were `libmpeos.a`, the file would be located in `%OCAPROOT%\gen%\%OCAPTC%\lib\libmpeos.a`.

- ◆ **NOTE:** The OCAP stack, including the JVM, uses additional common standard C-library functions typically supplied with most compiler environments. In general, the subset of C-library functions is limited to those provided via `stdio.h` (for example, `stdin`, `stdout`, and `stderr`), `string.h` (for example, `stccmp`, etc.), and `memory.h` (for example, `memcpy`, etc.). If for some reason any of the common C-library functions are not available, implementations for the functions should be coded and included within the MPEOS library or an additional library specific to the build environment for the target platform.

Understanding the implementation

The various MPEOS APIs are organized by functionality in an `mpe\os\target operating system` subdirectory for each platform.

If the target operating system does not directly support some of the OCAP functional requirements, the MPEOS APIs must supply code to complete the functional gaps. Part of the porting process is to determine which operating system APIs can be used and what additional functionality will have to be supplied within the MPEOS layer.

The porting process can be simplified by implementing the APIs in discrete stages with intermediate testing and debugging. There is no explicit ordering, nor are there advantages in completing and verifying the APIs one at a time. A team of programmers can implement many of the APIs concurrently if the port is undertaken in logical groupings.

One basic strategy for porting is to develop a time line based on when a particular function is needed within the entire porting process. The APIs targeted first would be the ones that must be in place for the JVM to operate. Then each of the next API categories pertain to the progressively more esoteric features of the system (for example, Digital Video Recorder (DVR)).

Basic operating-system services

The basic operating-system services include the following source files:

- `mpeos_dbg.c` includes simple functions for printing or logging `printf` compatible statements for debugging purposes. If you use `vprintf` and the native system does not supply thread-safety or atomic printing support, then the implementation may need to be enhanced later in the porting process with the use of mutexes for protection.
- `mpeos_dll.c` includes Dynamic Link Library (DLL) support including the ability to look up function addresses by name. If the target operating system does not directly support DLL symbol-lookup functionality, then these functions must supply this functionality. This can be done dynamically with Perl scripts if the build environment allows for conducting string searches within code components. It can also be supported statically by using a function registration table that is maintained by hand and statically compiled into the associated component libraries.
- `mpeos_event.c` includes general purpose event queue functions for supporting asynchronous notifications.

| | |
|-----------------------------|---|
| <code>mpeos_mem.c</code> | includes memory allocation functions, which include support for defining and segregating memory regions to assist with limiting memory fragmentation. If the target system directly supports handle-based memory allocations, then many of the memory functions should be fairly simple to implement by mapping to those handle-based functions. If the system does not support handle-based memory or memory types, implement all of the memory functions to use the same operating-system functions. There are no hard requirements for handle-based support. In fact, not all allocations should use handle-based memory where a dynamic lock must be set to access the memory. Allocations by the JVM in particular should never use memory that must be dynamically locked, unless the lock is performed at the MPEOS API allocation level and remains locked until it is returned by the JVM. |
| <code>mpeos_sync.c</code> | includes thread synchronization related functions (for example, mutexes and condition variables). |
| <code>mpeos_thread.c</code> | includes thread creation, maintenance, and status-related functions. The most significant requirements within the thread implementation are the JVM-related support features. Specifically, the OCAP stack requirements for all threads that will execute within the JVM realm are very strict. For example, the Esmertec JVM threads must have at a minimum a 64K stack size. |
| <code>mpeos_time.c</code> | includes simple functions for acquiring and converting the system time. |
| <code>mpeos_util.c</code> | includes general purpose functions supporting common process oriented features such as <code>setjmp</code> , <code>longjmp</code> , environment variables, etc. The environment variable support is used mostly for convenient runtime configuration of the OCAP stack and the JVM's execution mode. In particular, the environment is used to set up the JVM's startup properties. If the target system does not support environment variables, then the target implementation may need to rely on the use of a file from an available file system or possibly static compilation of the environment values into an internal static structure. |

DVR services

The DVR services include the following source files:

| | |
|--------------------------|---|
| <code>mpeos_dvr.c</code> | includes functions to create and play back digital video recordings. It also provides time-shift functionality that enables record, pause, rewind, and fast-forward of real-time broadcast content. |
|--------------------------|---|

File system and communication services

The file system and communications services include the following source files:

`mpeos_file.c` includes file system-related functions. These functions are designed to allow access to multiple file-system types. During initialization, the target-specific file systems are registered and subsequently, based on use of a Uniform Resource Locator (URL) syntax, the runtime file system functions are directed to the MPEOS file-system specific implementations. This allows for unified file functions at the MPE level. Each of the file-system specific implementations implements the same set of MPEOS functions and are typically organized within separate `mpeos_file-system type.c` files. File types include hard drive, Read-Only Memory (ROM), Non-Volatile Random Access Memory (NVRAM), etc.

`mpeos_socket.c`

includes Berkeley System Distribution (BSD) style socket support.

`mpeos_storage.c`

includes support for notifying registered listeners when storage devices are added, removed, or change states. When the Storage Manager is first constructed, it must register as a listener with the OCAP Event Dispatch Manager so that it can be notified of such events.

`mpeos_uievent.c`

includes input device event support for Java's Abstract Windowing Toolkit (AWT) input event requirements. The implementation must support a synchronous-blocking strategy to fulfill the requirements of an AWT Java thread.

Graphics services

The graphic services include the following source files:

`mpeos_disp.c` includes elementary graphical display operations. The set of functions supported at the MPEOS layer should be robust enough to support any middleware graphics system.

Media services

The media services include the following source files:

`mpeos_caption.c`

includes support for displaying captions encoded in analog and digital video streams.

`mpeos_filter.c`

includes support to acquire a section, or sections, from a Moving Picture Experts Group (MPEG) table section source that meets the criteria expressed in the form of a Packet Identifier (PID) and a filtering specification.

| | |
|----------------------------|---|
| <code>mpeos_media.c</code> | includes Moving Picture Experts Group (MPEG) transport stream tuning, decoding, and display configuration related functions. |
| <code>mpeos_snd.c</code> | includes support for creating and deleting players, discovering devices that can play a sound data MIME type, and controlling playback of sound data. |
| <code>mpeos_si.c</code> | includes MPEG service-information maintenance and support functions. |

POD services

This implementation uses the term Point-Of-Deployment (POD) to reference removable security, also known at OpenCable as a CableCARD. POD services includes the following source files:

| | |
|--------------------------|--|
| <code>mpeos_cd1.c</code> | includes support for managing the OpenCable download process. Common download is used to download new code to the set-top box through the CableCARD. |
| <code>mpeos_pod.c</code> | includes POD access services. If the target platform does not directly support a POD device, you can statically or dynamically implement these functions to emulate the functionality required by the OCAP stack for POD services. |

Miscellaneous

The miscellaneous services includes the following source files:

| | |
|---------------------------------|---|
| <code>mpeos_frontpanel.c</code> | includes support for an Xlet with <code>MonitorAppPermission</code> to control the front panel indicators and text display of a set-top box. |
| <code>mpeos_main.c</code> | includes the main process-related interface support for launching the OCAP stack and JVM within a single process environment (for example, process and address space). <code>mpeos_main.c</code> should contain the analogous standard C-library <code>main()</code> routine that functions as the entry point to the process. The actual specifics of the <code>main()</code> function entry point are completely platform specific, but the activities that must be performed to initialize and startup the OCAP stack are well defined. The main support functionality that must be supplied by the process entry point code is as follows: <ol style="list-style-type: none"> 1. Call any MPEOS API implementation-specific initialization functions. This is referred to as low-level initialization within the OCAP stack. |

2. Initialize the platform-independent MPE portion of the OCAP stack. This is accomplished by calling the `mpe_sysInit()` function. Depending on how the actual native portions of the OCAP stack are organized and packaged for a particular target platform, you may need to perform additional build steps to make `mpe_sysInit()` accessible from the MPE API portion of the code. For example, if the MPE portion of the stack was built into a separate DLL rather than the MPEOS portion, then additional effort would likely be necessary to make the `mpe_sysInit()` function accessible from the MPE API DLL. To avoid this interdependency issue, you should build the MPE and MPEOS libraries into a single platform module.
3. Initialize the JVM portion of the OCAP stack. This initialization step is performed by calling the MPE layer `mpe_jvmCreateVM()` function, which must be exposed in the MPE layer as previously described.
4. Invoke the initial Java class of the OCAP stack. This step launches the final initialization phase of the Java portion of the OCAP stack, which can subsequently result in the launching of any out-of-band or in-band OCAP Java applications that may be signaled. Startup of the initial Java class is performed by calling, in the same manner as described above, the `mpe_jvmExecuteMain()` function.

Operating system-specific header files

Along with the MPEOS header files, there are requirements for particular operating system header files. These header files are used to organize associated implementation types and structure definitions local to the operating system APIs. These header files also supply type definition bindings for `mpe_` types exposed above the MPEOS API level that must resolve to platform-specific types. The convention for organizing these associated definitions is to place them within like-named header files located within a local include directory (such as `mpe\os\targetOS\include`). For example, the file `os_thread.h` contains various macros and type definitions that bind to platform-specific values along with internal implementation structures used to maintain threads created for the upper layers of the OCAP stack (MPE and JVM).

The operating-system specific headers files are the base set of target-specific header files that must be provided by a target port in order for other portions of the OCAP stack to build correctly. Currently, the operating-system specific header files that must be made available to the upper portions of the OCAP stack are:

| | |
|--------------------------|---------------------------|
| <code>os_config.h</code> | <code>os_dll.h</code> |
| <code>os_dvr.h</code> | <code>os_error.h</code> |
| <code>os_event.h</code> | <code>os_file.h</code> |
| <code>os_gfx.h</code> | <code>os_media.h</code> |
| <code>os_socket.h</code> | <code>os_storage.h</code> |
| <code>os_sync.h</code> | <code>os_thread.h</code> |
| <code>os_time.h</code> | <code>os_types.h</code> |
| <code>os_util.h</code> | |

- ◆ **NOTE:** Platform-specific header files (`platform_xxx.h`) are port-specific and may have customized numbers and names for each port.

Configuration

The file `os_config.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain the configuration.

Definitions

The platform-specific configuration definitions are:

`OS_NAME` specifies the operating system name.

Example syntax:

```
#define OS_NAME "PowerTV"
```

Example MPE mapping syntax:

```
MPE.SYS.OSNAME
```

Type definitions

The platform-specific configuration type definitions are:

none

Dynamic link library (DLL) The file os_dll.h contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to support DLLs.

Definitions The platform-specific DLL definitions are:

none

Type definitions The platform-specific DLL type definition is:

`os_Dlmod` and `os_DlmodData`

`os_Dlmod` specifies a handle to an `os_DlmodData` structure.
`os_DlmodData` specifies the DLL handle structure.

Example syntax:

```
typedef struct _os_DLMod {
    void *dll_handle;
    os_SymbolTbl *dll_symbols;
} *os_Dlmod, os_DlmodData;
```

Example MPE mapping syntax:

```
typedef os_Dlmod mpe_Dlmod;
typedef os_DlmodData mpe_DlmodData;
```

Digital Video Recorder (DVR) The file os_dvr.h contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain DVR.

Definitions The platform-specific DVR definitions are:

`OS_DVR_MAX_NAME_SIZE`

specifies the maximum length of a recording name.

Example syntax:

```
#define OS_DVR_MAX_NAME_SIZE 32
```

Example MPE mapping syntax:

```
#define MPE_DVR_MAX_NAME_SIZE OS_DVR_MAX_NAME_SIZE
```

`OS_DVR_MEDIA_VOL_MAX_PATH_SIZE`

specifies the maximum length of a recording name.

Example syntax:

```
#define OS_DVR_MEDIA_VOL_MAX_PATH_SIZE
    OS_FS_MAX_PATH
```

Example MPE mapping syntax:

```
#define MPE_DVR_MEDIA_VOL_MAX_PATH_SIZE
    OS_DVR_MEDIA_VOL_MAX_PATH_SIZE
```

`OS_DVR_POSITIVE_INFINITY`
specifies a value for jumping to the end of a record during playback.

Example syntax:

```
#define OS_DVR_POSITIVE_INFINITY
    0x7FFFFFFFFFFFFFFFLL
```

Example MPE mapping syntax:

```
#define MPE_DVR_POSITIVE_INFINITY
    OS_DVR_POSITIVE_INFINITY
```

Type definitions The platform-specific DVR type definitions are:

`os_MediaVolumeInfo`
specifies an opaque data type that represents a media storage volume.

Example syntax:

```
typedef struct _os_MediaVolumeInfo {
    char vol_path
        [OS_DVR_MEDIA_VOL_MAX_PATH_SIZE + 1];
    uint64_t vol_size;
    ui32 device_id;
    os_FreeSpaceAlarmStatus
        alarmStatus[MAX_MEDIA_VOL_LEVEL];
} os_MediaVolumeInfo;
```

Example MPE mapping syntax:

```
typedef os_MediaVolumeInfo mpeos_MediaVolume;
```

Error codes

The file `os_error.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain error codes. Normally, these values are defined to match an existing operating system error codes. For example, on POSIX compatible systems the memory allocation error `OS_ENOMEM` would simply be defined to `ENOMEM`. In some cases, some values may not have corresponding operating system values, in which case an unused value for the target platform should be selected as the representative value.

Definitions

The platform-specific error code definitions are:

`OS_ERRBASE` defines a common base of error code for the platform.

Example syntax:

```
#define OS_ERRBASE (100)
```

| | |
|------------|---|
| OS_SUCCESS | indicates a successful completion of an mpe_ or mpeos_ function call. This is the value that should be compared against to determine the API completion status. Implementation code should not assume any specific value (for example, zero). |
| | Example syntax: <code>#define OS_SUCCESS (0)</code> |
| | Example MPE mapping syntax: <code>#define MPE_SUCCESS OS_SUCCESS</code> |
| OS_EINVAL | indicates at least one input parameter to the function has an invalid parameter value. |
| | Example syntax: <code>#define OS_EINVAL (OS_ERRBASE+1)</code> |
| | Example MPE mapping syntax: <code>#define MPE_EINVAL OS_EINVAL</code> |
| OS_ENOMEM | indicates the function failed due to insufficient memory resource availability. |
| | Example syntax: <code>#define OS_ENOMEM (OS_ERRBASE+2)</code> |
| | Example MPE mapping syntax: <code>#define MPE_ENOMEM OS_ENOMEM</code> |
| OS_EBUSY | indicates the device or resource is busy. |
| | Example syntax: <code>#define OS_EBUSY (OS_ERRBASE+3)</code> |
| | Example MPE mapping syntax: <code>#define MPE_EBUSY OS_EBUSY</code> |
| OS_EMUTEX | indicates a function failure for creation/deletion/acquisition of a mutex. |
| | Example syntax: <code>#define OS_EMUTEX (OS_ERRBASE+4)</code> |
| | Example MPE mapping syntax: <code>#define MPE_EMUTEX OS_EMUTEX</code> |
| OS_ECOND | indicates a function failure for creation/deletion/acquisition of a condition object. |
| | Example syntax: <code>#define OS_ECOND (OS_ERRBASE+5)</code> |
| | Example MPE mapping syntax: <code>#define MPE_ECOND OS_ECOND</code> |

| | |
|----------------|--|
| OS_EEVENT | indicates the function failed. Example syntax: <code>#define OS_EEVENT (OS_ERRBASE+6)</code> |
| OS_ENODATA | indicates the resource/queue has no data. Example syntax: <code>#define OS_ENODATA (OS_ERRBASE+7)</code> |
| OSETIMEOUT | indicates the function has returned due to a time-out condition. Example syntax: <code>#define OSETIMEOUT (OS_ERRBASE+8)</code> |
| OSETHREADDEATH | indicates the thread in which this function is executing has been marked for death. Example syntax: <code>#define OSETHREADDEATH (OS_ERRBASE+9)</code> |

Type definitions

The platform-specific error code type definitions are:

none

Event

The file `os_event.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain events.

Definitions

The platform-specific event definitions are:

none

| | |
|-------------------------|---|
| Type definitions | The platform-specific event type definitions are: os_Event specifies the event type definitions. Example syntax: typedef ui32 os_Event; Example MPE mapping syntax: typedef os_Event mpe_Event; os_EventQueue resolves the platform-specific event queue type. Example syntax: typedef Pk_Queue *os_EventQueue; Example MPE mapping syntax: typedef os_EventQueue mpe_EventQueue; |
|-------------------------|---|

File system

The file `os_file.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain the file system.

Definitions

The platform-specific file system definitions are:

`OS_FS_MAX_PATH` specifies the maximum path length for the file system.

Example syntax:

```
#define OS_FS_MAX_PATH 1024
```

Example MPE mapping syntax:

```
#define MPE_FS_MAX_PATH OS_FS_MAX_PATH
```

`OS_FS_SEPARATION_STRING`

specifies the string separation character.

Example syntax:

```
#define OS_FS_SEPARATION_STRING "/"
```

Example MPE mapping syntax:

```
#define MPE_FS_SEPARATION_STRING  
    OS_FS_SEPARATION_STRING
```

`OS_FS_DEFAULT_SYS_DIR`

specifies the default system directory.

Example syntax:

```
#define OS_FS_DEFAULT_SYS_DIR ".."
```

Example MPE mapping syntax:

```
#define MPE_FS_DEFAULT_SYS_DIR  
    OS_FS_DEFAULT_SYS_DIR
```

`OS_FS_MAX_MOUNT_POINT_SIZE`
 specifies the maximum length in points of the virtual mount
 for the MPE, file-system drivers (for example, /itfs)
 excluding the null terminator.

Example syntax:

```
#define OS_FS_MAX_MOUNT_POINT_SIZE 10
```

Example MPE mapping syntax:

```
#define MPE_FS_MAX_MOUNT_POINT_SIZE  
OS_FS_MAX_MOUNT_POINT_SIZE
```

Type definitions The platform-specific file system type definitions are:
 none

Graphics The file `os_gfx.h` contains mandatory definitions, data types and structures
 that bind to platform-specific values, and internal implementation
 structures used to maintain graphics.

Definitions The platform-specific graphics definitions are:
 none

Type definitions The platform-specific graphics type definitions are:
`os_GfxContext` specifies the DirectFB sub-surface.

Example syntax:

```
typedef struct os_GfxContext {  
    IDirectFBSurface *subs;  
} os_GfxContext;
```

`os_GfxSurface` specifies the a DirectFB surface.

Example syntax:

```
typedef struct os_GfxSurface {  
    IDirectFBSurface *os_s;  
} os_GfxSurface;
```

`os_GfxFont` specifies a DirectFB graphics font.

Example syntax:

```
typedef struct os_GfxFont {  
    IDirectFBFont *dfb_fnt;  
} os_GfxFont;
```

`os_GfxScreen` specifies the a DirectFB entry point.

Example syntax:

```
typedef struct os_GfxScreen {  
    IDirectFB *dfb;  
} os_GfxScreen;
```

Media

The file `os_media.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to support media.

Definitions

The platform-specific media definitions are:

`OS_MEDIA_ERROR_BADTUNINGREQUEST`
specifies the tune request failed.

Example syntax:

```
#define OS_MEDIA_ERROR_BADTUNINGREQUEST  
    kTv_BadTuningRequestErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_BAD_TUNING_REQUEST  
= OS_MEDIA_ERROR_BADTUNINGREQUEST
```

`OS_MEDIA_ERROR_INVALIDID`
specifies an invalid tuner identifier.

Example syntax:

```
#define OS_MEDIA_ERROR_INVALIDID kTv_InvalidIDErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_INVALID_ID  
= OS_MEDIA_ERROR_INVALIDID
```

`OS_MEDIA_ERROR_INVALIDCHANNEL`
specifies an invalid channel.

Example syntax:

```
#define OS_MEDIA_ERROR_INVALIDCHANNEL  
    kTv_InvalidChannelErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_INVALID_CHANNEL  
= OS_MEDIA_ERROR_INVALIDCHANNEL
```

`OS_MEDIA_ERROR_INVALIDSOURCEID`
specifies an invalid source identifier.

Example syntax:

```
#define OS_MEDIA_ERROR_INVALIDSOURCEID  
    kTv_InvalidSourceIDErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_INVALID_SOURCEID  
= OS_MEDIA_ERROR_INVALIDSOURCEID
```

`OS_MEDIA_ERROR_INVALIDPLAYER`
specifies an invalid player.

Example syntax:

```
#define OS_MEDIA_ERROR_INVALIDPLAYER (1)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_INVALID_PLAYER  
= OS_MEDIA_ERROR_INVALIDPLAYER
```

`OS_MEDIA_ERROR_NOLONGERAUTHORIZED`
specifies a time-out error occurred on the authorization.

Example syntax:

```
#define OS_MEDIA_ERROR_NOLONGERAUTHORIZED  
KTv_NoLongerAuthorizedErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_NO_LONGER_AUTHORIZED  
= OS_MEDIA_ERROR_NOLONGERAUTHORIZED
```

`OS_MEDIA_ERROR_AUTHORIZATIONFAILED`
specifies the authorization failed.

Example syntax:

```
#define OS_MEDIA_ERROR_AUTHORIZATIONFAILED (2)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_AUTHORIZATION_FAILED  
= OS_MEDIA_ERROR_AUTHORIZATIONFAILED
```

`OS_MEDIA_ERROR_APINOTIMPLEMENTED`
specifies the API is not implemented.

Example syntax:

```
#define OS_MEDIA_ERROR_APINOTIMPLEMENTED (6)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_API_NOT_IMPLEMENTED  
= OS_MEDIA_ERROR_APINOTIMPLEMENTED
```

`OS_MEDIA_ERROR_APINOTSUPPORTED`
specifies the API is not supported by the target platform.

Example syntax:

```
#define OS_MEDIA_ERROR_APINOTSUPPORTED (7)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_API_NOT_SUPPORTED  
= OS_MEDIA_ERROR_APINOTSUPPORTED
```

OS_MEDIA_ERROR_GENERIC
specifies an error occurred in the operating system.

Example syntax:

```
#define OS_MEDIA_ERROR_GENERIC (8)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_OS = OS_MEDIA_ERROR_GENERIC
```

OS_MEDIA_ERROR_STREAMOPEN
specifies the stream is open.

Example syntax:

```
#define OS_MEDIA_ERROR_STREAMOPEN (3)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_STREAM_OPEN = OS_MEDIA_ERROR_STREAMOPEN
```

OS_MEDIA_ERROR_STREAMREAD
specifies the is being read.

Example syntax:

```
#define OS_MEDIA_ERROR_STREAMREAD (4)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_STREAM_READ = OS_MEDIA_ERROR_STREAMREAD
```

OS_MEDIA_ERROR_BUFFEROVERRUN
specifies the buffer is full.

Example syntax:

```
#define OS_MEDIA_ERROR_BUFFEROVERRUN (5)
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_BUFFER_OVERRUN  
= OS_MEDIA_ERROR_BUFFEROVERRUN
```

OS_MEDIA_ERROR_NOIDSAVAILABLE
specifies the identifiers are not available.

Example syntax:

```
#define OS_MEDIA_ERROR_NOIDSAVAILABLE  
KTv_NoIDsAvailableErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_NO_IDS_AVAILABLE  
= OS_MEDIA_ERROR_NOIDSAVAILABLE
```

OS_MEDIA_ERROR_PRIORITYLEVEL

specifies the priority level is not acceptable.

Example syntax:

```
#define OS_MEDIA_ERROR_PRIORITYLEVEL
    kTv_PriorityLevelErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_PRIORITY_LEVEL
= OS_MEDIA_ERROR_PRIORITYLEVEL
```

OS_MEDIA_ERROR_RESOURCENOTACTIVE

specifies the resource is not active.

Example syntax:

```
#define OS_MEDIA_ERROR_RESOURCENOTACTIVE
    kTv_ResourceNotActiveErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_RESOURCE_NOT_ACTIVE
= OS_MEDIA_ERROR_RESOURCENOTACTIVE
```

OS_MEDIA_ERROR_NOTOWNER

specifies an invalid owner.

Example syntax:

```
#define OS_MEDIA_ERROR_NOTOWNER
    kTv_NotOwnerErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_NOT_OWNER = OS_MEDIA_ERROR_NOTOWNER
```

OS_MEDIA_ERROR_BADLINK

specifies a failed link.

Example syntax:

```
#define OS_MEDIA_ERROR_BADLINK kTv_BadLinkErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_BAD_LINK = OS_MEDIA_ERROR_BADLINK
```

OS_MEDIA_ERROR_BLACKEDOUT

specifies a black out.

Example syntax:

```
#define OS_MEDIA_ERROR_BLACKEDOUT kTv_BlackedOutErr
```

Example MPE mapping syntax:

```
MPE_ERROR_MEDIA_BLACKED_OUT=OS_MEDIA_ERROR_BLACKEDOUT
```

| | |
|-------------------------|---|
| | <p><code>OS_MEDIA_ERROR_ECMSTREAM</code> specifies an Entitlement Control Messages (ECM) stream.</p> <p>Example syntax:</p> <pre>#define OS_MEDIA_ERROR_ECMSTREAM kTv_ECMStreamErr</pre> <p>Example MPE mapping syntax:</p> <pre>MPE_ERROR_MEDIA_ECM_STREAM=OS_MEDIA_ERROR_ECMSTREAM</pre> |
| Type definitions | <p>The platform-specific media type definition is: <code>none</code></p> |
| Networking | <p>The file <code>os_socket.h</code> contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain networking.</p> |
| Definitions | <p>The platform-specific networking definitions are:</p> <p><code>OS_SOCKET_FD_SETSIZE</code> specifies the maximum number of file descriptors represented in the <code>mpe_SocketFDSet</code> data type.</p> <p>Example syntax:</p> <pre>#define OS_SOCKET_FD_SETSIZE FD_SETSIZE</pre> <p>Example MPE mapping syntax:</p> <pre>#define MPE_SOCKET_FD_SETSIZE OS_SOCKET_FD_SETSIZE</pre> <p><code>OS_SOCKET_MAXHOSTNAMELEN</code> specifies the maximum length of a host name in bytes, including the terminating NULL byte.</p> <p>Example syntax:</p> <pre>#define OS_SOCKET_MAXHOSTNAMELEN MAXHOSTNAMELEN</pre> <p>Example MPE mapping syntax:</p> <pre>#define MPE_SOCKET_MAXHOSTNAMELEN OS_SOCKET_MAXHOSTNAMELEN</pre> <p><code>OS_SOCKET_MSG_00B</code> sends or receives out-of-band data on sockets that support out-of-band data.</p> <p>Example syntax:</p> <pre>#define OS_SOCKET_MSG_00B MSG_00B</pre> <p>Example MPE mapping syntax:</p> <pre>#define MPE_SOCKET_MSG_00B OS_SOCKET_MSG_00B</pre> |

OS_SOCKET_MSG_PEEK
 peeks at an incoming message.

Example syntax:

```
#define OS_SOCKET_MSG_PEEK MSG_PEEK
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_MSG_PEEK OS_SOCKET_MSG_PEEK
```

OS_SOCKET_SHUTDOWN_RD
 is used to disable further receive operations.

Example syntax:

```
#define OS_SOCKET_SHUTDOWN_RD 0
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SHUT_RD OS_SOCKET_SHUTDOWN_RD
```

OS_SOCKET_SHUTDOWN_RDWR
 is used to disable further send and receive operations.

Example syntax:

```
#define OS_SOCKET_SHUTDOWN_RDWR 2
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SHUT_RDWR OS_SOCKET_SHUTDOWN_RDWR
```

OS_SOCKET_SHUTDOWN_WR
 is used to disable further send operations.

Example syntax:

```
#define OS_SOCKET_SHUTDOWN_WR 1
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SHUT_WR OS_SOCKET_SHUTDOWN_WR
```

OS_SOCKET_DGRAM
 specifies a datagram-based socket.

Example syntax:

```
#define OS_SOCKET_DGRAM SOCK_DGRAM
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_DGRAM OS_SOCKET_DGRAM
```

OS_SOCKET_STREAM
 specifies a stream-based socket.

Example syntax:

```
#define OS_SOCKET_STREAM SOCK_STREAM
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_STREAM OS_SOCKET_STREAM
```

OS_SOCKET_INVALID_SOCKET
indicates an mpe_Socket descriptor is not valid.

Example syntax:
`#define OS_SOCKET_INVALID_SOCKET -1`

Example MPE mapping syntax:
`#define MPE_SOCKET_INVALID_SOCKET OS_SOCKET_INVALID_SOCKET`

OS_SOCKET_FIONBIO
is used to clear or turn on the non-blocking flag for the socket.

Example syntax:
`#define OS_SOCKET_FIONBIO FIONBIO`

Example MPE mapping syntax:
`#define MPE_SOCKET_FIONBIO OS_SOCKET_FIONBIO`

OS_SOCKET_FIONREAD
is used to return the number of bytes currently in the receive buffer for the socket.

Example syntax:
`#define OS_SOCKET_FIONREAD FIONREAD`

Example MPE mapping syntax:
`#define MPE_SOCKET_FIONREAD OS_SOCKET_FIONREAD`

OS_SOCKET_AF_INET4
specifies the address family for IPv4 sockets.

Example syntax:
`#define OS_SOCKET_AF_INET4 AF_INET`

Example MPE mapping syntax:
`#define MPE_SOCKET_AF_INET4 OS_SOCKET_AF_INET4`

OS_SOCKET_IN4ADDR_ANY
specifies the wildcard IPv4 address (matches any address).

Example syntax:
`#define OS_SOCKET_IN4ADDR_ANY INADDR_ANY`

Example MPE mapping syntax:
`#define MPE_SOCKET_IN4ADDR_ANY OS_SOCKET_IN4ADDR_ANY`

OS_SOCKET_IN4ADDR_LOOPBACK

specifies the wildcard IPv4 loopback address. (For more information, refer to the POSIX definition of INADDR_LOOPBACK.) The loopback address value depends on the byte-ordering of the target system. Either MPE_BIG_ENDIAN or MPE_LITTLE_ENDIAN must be defined.

Example syntax:

```
#ifdef MPE_BIG_ENDIAN
#define OS_SOCKET_IN4ADDR_LOOPBACK 0x0100007f
#endif
#ifndef MPE_LITTLE_ENDIAN:
#define OS_SOCKET_IN4ADDR_LOOPBACK 0x7f000001
#endif
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_IN4ADDR_LOOPBACK OS_SOCKET_IN4ADDR_LOOPBACK
```

OS_SOCKET_INET4_ADDRSTRLEN

specifies the maximum length of an IPv4 address string (includes the null terminator). (For more information, refer to the POSIX definition of INET_ADDRSTRLEN.)

Example syntax:

```
#define OS_SOCKET_INET4_ADDRSTRLEN 16
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_INET4_ADDRSTRLEN OS_SOCKET_INET4_ADDRSTRLEN
```

OS_SOCKET_SOL_SOCKET

specifies a socket level option.

Example syntax:

```
#define OS_SOCKET_SOL_SOCKET SOL_SOCKET
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SOL_SOCKET OS_SOCKET_SOL_SOCKET
```

OS_SOCKET_SO_BROADCAST

controls whether transmission of broadcast messages is supported by the protocol.

Example syntax:

```
#define OS_SOCKET_SO_BROADCAST SO_BROADCAST
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_BROADCAST OS_SOCKET_SO_BROADCAST
```

OS_SOCKET_SO_DEBUG
controls whether debugging information is being recorded.

Example syntax:

```
#define OS_SOCKET_SO_DEBUG SO_DEBUG
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_DEBUG OS_SOCKET_SO_DEBUG
```

OS_SOCKET_SO_DONTROUTE
controls whether outgoing messages bypass the standard routing facilities.

Example syntax:

```
#define OS_SOCKET_SO_DONTROUTE SO_DONTROUTE
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_DONTROUTE OS_SOCKET_SO_DONTROUTE
```

OS_SOCKET_SO_ERROR
reports information about error status.

Example syntax:

```
#define OS_SOCKET_SO_ERROR SO_ERROR
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_ERROR OS_SOCKET_SO_ERROR
```

OS_SOCKET_SO_KEEPALIVE
controls whether connections are kept active with periodic transmission of messages, if the protocol is supported.

Example syntax:

```
#define OS_SOCKET_SO_KEEPALIVE SO_KEEPALIVE
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_KEEPALIVE OS_SOCKET_SO_KEEPALIVE
```

OS_SOCKET_SO_LINGER
controls whether the socket lingers on if data is present.

Example syntax:

```
#define OS_SOCKET_SO_LINGER SO_LINGER
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_LINGER OS_SOCKET_SO_LINGER
```

OS_SOCKET_SO_OOBINLINE
 controls whether the socket leaves received out-of-band data (data marked urgent) in-line.

Example syntax:

```
#define OS_SOCKET_SO_OOBINLINE SO_OOBINLINE
```

Example MPE mapping syntax:

```
#defineMPE_SOCKET_SO_OOBINLINEOS_SOCKET_SO_OOBINLINE
```

OS_SOCKET_SO_RCVBUF
 controls the size of the receive buffer.

Example syntax:

```
#define OS_SOCKET_SO_RCVBUF SO_RCVBUF
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_RCVBUF OS_SOCKET_SO_RCVBUF
```

OS_SOCKET_SO_RCVLOWAT
 controls the minimum number of bytes (or low-water mark value) to process for socket input operations.

Example syntax:

```
#define OS_SOCKET_SO_RCVLOWAT SO_RCVLOWAT
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_RCVLOWAT OS_SOCKET_SO_RCVLOWAT
```

OS_SOCKET_SO_RCVTIMEO
 controls the time-out value for input operations.

Example syntax:

```
#define OS_SOCKET_SO_RCVTIMEO SO_RCVTIMEO
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_RCVTIMEO OS_SOCKET_SO_RCVTIMEO
```

OS_SOCKET_SO_REUSEADDR
 controls whether the rules used in validating addresses should allow reuse of local addresses, if the protocol is supported.

Example syntax:

```
#define OS_SOCKET_SO_REUSEADDR SO_REUSEADDR
```

Example MPE mapping syntax:

```
#defineMPE_SOCKET_SO_REUSEADDROS_SOCKET_SO_REUSEADDR
```

OS_SOCKET_SO_SNDBUF

controls the send buffer size.

Example syntax:

```
#define OS_SOCKET_SO_SNDBUF SO_SNDBUF
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_SNDBUF OS_SOCKET_SO_SNDBUF
```

OS_SOCKET_SO SNDLOWAT

controls the minimum number of bytes (or low-water mark value) to process for socket output operations.

Example syntax:

```
#define OS_SOCKET_SO SNDLOWAT SO_SNDFLWAT
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO SNDLOWAT OS_SOCKET_SO SNDLOWAT
```

OS_SOCKET_SO SNDTIMEO

controls the time-out value specifying the amount of time an output function blocks due to flow control preventing data from being sent.

Example syntax:

```
#define OS_SOCKET_SO SNDTIMEO SO_SNDFLWAT
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO SNDTIMEO OS_SOCKET_SO SNDTIMEO
```

OS_SOCKET_SO_TYPE

reports the socket type.

Example syntax:

```
#define OS_SOCKET_SO_TYPE SO_TYPE
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_SO_TYPE OS_SOCKET_SO_TYPE
```

OS_SOCKET IPPROTO_IPV4

specifies the IPv4 protocol.

Example syntax:

```
#define OS_SOCKET IPPROTO_IPV4 IPPROTO_IP
```

Example MPE mapping syntax:

```
#define MPE_SOCKET IPPROTO_IPV4 OS_SOCKET IPPROTO_IP
```

OS_SOCKET IPPROTO_TCP

specifies the Transmission Control Protocol/Internet Protocol (TCP/IP).

Example syntax:

```
#define OS_SOCKET IPPROTO_TCP IPPROTO_TCP
```

Example MPE mapping syntax:

```
#define MPE_SOCKET IPPROTO_TCP OS_SOCKET IPPROTO_TCP
```

OS_SOCKET IPPROTO_UDP

specifies the User Datagram Protocol/Internet Protocol (UDP/IP).

Example syntax:

```
#define OS_SOCKET IPPROTO_UDP IPPROTO_UDP
```

Example MPE mapping syntax:

```
#define MPE_SOCKET IPPROTO_UDP OS_SOCKET IPPROTO_UDP
```

OS_SOCKET IPV4_ADD_MEMBERSHIP

joins a multicast group on a specified local interface.

Example syntax:

```
#define OS_SOCKET IPV4_ADD_MEMBERSHIP IP_ADD_MEMBERSHIP
```

Example MPE mapping syntax:

```
#define MPE_SOCKET IPV4_ADD_MEMBERSHIP  
OS_SOCKET IPV4_ADD_MEMBERSHIP
```

OS_SOCKET IPV4_DROP_MEMBERSHIP

leaves a multicast group.

Example syntax:

```
#define OS_SOCKET IPV4_DROP_MEMBERSHIP IP_DROP_MEMBERSHIP
```

Example MPE mapping syntax:

```
#define MPE_SOCKET IPV4_DROP_MEMBERSHIP  
OS_SOCKET IPV4_DROP_MEMBERSHIP
```

OS_SOCKET IPV4_MULTICAST_IF

specifies the interface for outgoing multicast datagrams sent on this socket.

Example syntax:

```
#define OS_SOCKET IPV4_MULTICAST_IF IP_MULTICAST_IF
```

Example MPE mapping syntax:

```
#define MPE_SOCKET IPV4_MULTICAST_IF  
OS_SOCKET IPV4_MULTICAST_IF
```

OS_SOCKET_IPV4_MULTICAST_LOOP
enables or disables the local loopback of multicast datagrams.

Example syntax:

```
#define OS_SOCKET_IPV4_MULTICAST_LOOP IP_MULTICAST_LOOP
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_IPV4_MULTICAST_LOOP  
OS_SOCKET_IPV4_MULTICAST_LOOP
```

OS_SOCKET_IPV4_MULTICAST_TTL
sets or reads the time-to-live value of outgoing multicast datagrams for this socket.

Example syntax:

```
#define OS_SOCKET_IPV4_MULTICAST_TTL IP_MULTICAST_TTL
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_IPV4_MULTICAST_TTL  
OS_SOCKET_IPV4_MULTICAST_TTL
```

OS_SOCKET_TCP_NODELAY
disables TCP's Nagle algorithm, when set.

Example syntax:

```
#define OS_SOCKET_TCP_NODELAY TCP_NODELAY
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_TCP_NODELAY OS_SOCKET_TCP_NODELAY
```

OS_MPE_SOCKET_EACCES
indicates access was denied due to insufficient privileges.

Example syntax:

```
#define OS_MPE_SOCKET_EACCES EACCES
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EACCES OS_MPE_SOCKET_EACCES
```

OS_MPE_SOCKET_EADDRINUSE
indicates the specified address is already in use.

Example syntax:

```
#define OS_MPE_SOCKET_EADDRINUSE EADDRINUSE
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EADDRINUSE OS_MPE_SOCKET_EADDRINUSE
```

OS_MPE_SOCKET_EADDRNOTAVAIL

indicates the specified address is not available from the local machine.

Example syntax:

```
#define OS_MPE_SOCKET_EADDRNOTAVAIL EADDRNOTAVAIL
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EADDRNOTAVAIL OS_MPE_SOCKET_EADDRNOTAVAIL
```

OS_MPE_SOCKET_EAFNOSUPPORT

indicates the specified address is not supported.

Example syntax:

```
#define OS_MPE_SOCKET_EAFNOSUPPORT EAFNOSUPPORT
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EAFNOSUPPORT OS_MPE_SOCKET_EAFNOSUPPORT
```

OS_MPE_SOCKET_EAGAIN

specifies the operation timed out and should be tried again.

Example syntax:

```
#define OS_MPE_SOCKET_EAGAIN EAGAIN
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EAGAIN OS_MPE_SOCKET_EAGAIN
```

OS_MPE_SOCKET_EALREADY

indicates a connection request is already in progress for the specified socket.

Example syntax:

```
#define OS_MPE_SOCKET_EALREADY EALREADY
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EALREADY OS_MPE_SOCKET_EALREADY
```

OS_MPE_SOCKET_EBADF

indicates an invalid file descriptor.

Example syntax:

```
#define OS_MPE_SOCKET_EBADF EBADF
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EBADF OS_MPE_SOCKET_EBADF
```

OS_MPE_SOCKET_ECONNABORTED

indicates a connection has been aborted.

Example syntax:

```
#define OS_MPE_SOCKET_ECONNABORTED ECONNABORTED
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ECONNABORTED OS_MPE_SOCKET_ECONNABORTED
```

OS_MPE_SOCKET_ECONNREFUSED
indicates the connection request was refused.

Example syntax:

```
#define OS_MPE_SOCKET_ECONNREFUSED ECONNREFUSED
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ECONNREFUSED OS_MPE_SOCKET_ECONNREFUSED
```

OS_MPE_SOCKET_ECONNRESET
indicates the connection was reset.

Example syntax:

```
#define OS_MPE_SOCKET_ECONNRESET ECONNRESET
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ECONNRESET OS_MPE_SOCKET_ECONNRESET
```

OS_MPE_SOCKET_EDESTADDRREQ
indicates an invalid destination address.

Example syntax:

```
#define OS_MPE_SOCKET_EDESTADDRREQ EDESTADDRREQ
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EDESTADDRREQ OS_MPE_SOCKET_EDESTADDRREQ
```

OS_MPE_SOCKET_EDOM
indicates an invalid field value.

Example syntax:

```
#define OS_MPE_SOCKET_EDOM EDOM
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EDOM OS_MPE_SOCKET_EDOM
```

OS_MPE_SOCKET_EHOSTNOTFOUND
indicates the host is unknown.

Example syntax:

```
#define OS_MPE_SOCKET_EHOSTNOTFOUND -1
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EHOSTNOTFOUND OS_MPE_SOCKET_EHOSTNOTFOUND
```

OS_MPE_SOCKET_EHOSTUNREACH
indicates the host cannot be reached.

Example syntax:

```
#define OS_MPE_SOCKET_EHOSTUNREACH EHOSTUNREACH
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EHOSTUNREACH OS_MPE_SOCKET_EHOSTUNREACH
```

OS_MPE_SOCKET_EINTR

indicates a signal interrupted the function before it could complete the requested operation.

Example syntax:

```
#define OS_MPE_SOCKET_EINTR EINTR
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EINTR OS_MPE_SOCKET_EINTR
```

OS_MPE_SOCKET_EIO

indicates an input/output (I/O) error occurred.

Example syntax:

```
#define OS_MPE_SOCKET_EIO EIO
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EIO OS_MPE_SOCKET_EIO
```

OS_MPE_SOCKET_EISCONN

indicates the socket is already connected.

Example syntax:

```
#define OS_MPE_SOCKET_EISCONN EISCONN
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EISCONN OS_MPE_SOCKET_EISCONN
```

OS_MPE_SOCKET_ELOOP

indicates more than the maximum number of loops occurred.

Example syntax:

```
#define OS_MPE_SOCKET_ELOOP ELOOP
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ELOOP OS_MPE_SOCKET_ELOOP
```

OS_MPE_SOCKET_EMFILE

indicates no more file descriptors are available for this process.

Example syntax:

```
#define OS_MPE_SOCKET_EMFILE EMFILE
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EMFILE OS_MPE_SOCKET_EMFILE
```

OS_MPE_SOCKET_EMSGSIZE
indicates the maximum message size was exceeded.

Example syntax:

```
#define OS_MPE_SOCKET_EMSGSIZE EMSGSIZE
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EMSGSIZE OS_MPE_SOCKET_EMSGSIZE
```

OS_MPE_SOCKET_ENAMETOOLONG
indicates the maximum path-name length was exceeded.

Example syntax:

```
#define OS_MPE_SOCKET_ENAMETOOLONG ENAMETOOLONG
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENAMETOOLONG OS_MPE_SOCKET_ENAMETOOLONG
```

OS_MPE_SOCKET_ENFILE
indicates no more file descriptors are available for the system.

Example syntax:

```
#define OS_MPE_SOCKET_ENFILE ENFILE
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENFILE OS_MPE_SOCKET_ENFILE
```

OS_MPE_SOCKET_ENETDOWN
indicates the local network interface used to reach the destination is down.

Example syntax:

```
#define OS_MPE_SOCKET_ENETDOWN ENETDOWN
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENETDOWN OS_MPE_SOCKET_ENETDOWN
```

OS_MPE_SOCKET_ENETUNREACH
indicates no route to the network is present.

Example syntax:

```
#define OS_MPE_SOCKET_ENETUNREACH ENETUNREACH
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENETUNREACH  
OS_MPE_SOCKET_ENETUNREACH
```

OS_MPE_SOCKET_ENOBUFS

indicates no buffer is present.

Example syntax:

```
#define OS_MPE_SOCKET_ENOBUFS ENOBUFS
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENOBUFS OS_MPE_SOCKET_ENOBUFS
```

OS_MPE_SOCKET_ENOPROTOOPT

indicates the option is not supported by the protocol.

Example syntax:

```
#define OS_MPE_SOCKET_ENOPROTOOPT ENOPROTOOPT
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENOPROTOOPT OS_MPE_SOCKET_ENOPROTOOPT
```

OS_MPE_SOCKET_ENORECOVERY

indicates an unexpected server failure occurred and cannot be recovered.

Example syntax:

```
#define OS_MPE_SOCKET_ENORECOVERY -1
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENORECOVERY OS_MPE_SOCKET_ENORECOVERY
```

OS_MPE_SOCKET_ENOSPC

indicates the size of the result buffer is inadequate.

Example syntax:

```
#define OS_MPE_SOCKET_ENOSPC NOSPC
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENOSPC OS_MPE_SOCKET_ENOSPC
```

OS_MPE_SOCKET_ENOTCONN

indicates the socket is not connected.

Example syntax:

```
#define OS_MPE_SOCKET_ENOTCONN ENOTCONN
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENOTCONN OS_MPE_SOCKET_ENOTCONN
```

OS_MPE_SOCKET_ENOTSOCK

indicates that a parameter does not refer to a socket.

Example syntax:

```
#define OS_MPE_SOCKET_ENOTSOCK ENOTSOCK
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ENOTSOCK OS_MPE_SOCKET_ENOTSOCK
```

OS_MPE_SOCKET_EOPNOTSUPP
indicates an unsupported operation.

Example syntax:
`#define OS_MPE_SOCKET_EOPNOTSUPP EOPNOTSUPP`

Example MPE mapping syntax:
`#defineMPE_SOCKET_EOPNOTSUPPOS_MPE_SOCKET_EOPNOTSUPP`

OS_MPE_SOCKET_EPIPE
indicates an invalid socket or that the socket is no longer connected.

Example syntax:
`#define OS_MPE_SOCKET_EPIPE EPIPE`

Example MPE mapping syntax:
`#define MPE_SOCKET_EPIPE OS_MPE_SOCKET_EPIPE`

OS_MPE_SOCKET_EPROTO
indicates an invalid protocol.

Example syntax:
`#define OS_MPE_SOCKET_EPROTO EPROTO`

Example MPE mapping syntax:
`#define MPE_SOCKET_EPROTO OS_MPE_SOCKET_EPROTO`

OS_MPE_SOCKET_EPROTONOSUPPORT
indicates an unsupported protocol.

Example syntax:
`#define OS_MPE_SOCKET_EPROTONOSUPPORT EPROTONOSUPPORT`

Example MPE mapping syntax:
`#define MPE_SOCKET_EPROTONOSUPPORT
OS_MPE_SOCKET_EPROTONOSUPPORT`

OS_MPE_SOCKET_EPROTOTYPE
indicates an incompatible protocol for the socket type.

Example syntax:
`#define OS_MPE_SOCKET_EPROTOTYPE EPROTOTYPE`

Example MPE mapping syntax:
`#defineMPE_SOCKET_EPROTOTYPEOS_MPE_SOCKET_EPROTOTYPE`

`OS_MPE_SOCKET_ETIMEDOUT`
 indicates a time-out occurred during the requested operation.

Example syntax:

```
#define OS_MPE_SOCKET_ETIMEDOUT ETIMEDOUT
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ETIMEDOUT OS_MPE_SOCKET_ETIMEDOUT
```

`OS_MPE_SOCKET_ETRYAGAIN`
 indicates a temporary and possibly transient error occurred, such as a failure of a server to respond.

Example syntax:

```
#define OS_MPE_SOCKET_ETRYAGAIN -1
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_ETRYAGAIN OS_MPE_SOCKET_ETRYAGAIN
```

`OS_MPE_SOCKET_EWOULDBLOCK`
 indicates the operation would block if attempted.

Example syntax:

```
#define OS_MPE_SOCKET_EWOULDBLOCK EWOULDBLOCK
```

Example MPE mapping syntax:

```
#define MPE_SOCKET_EWOULDBLOCK  
OS_MPE_SOCKET_EWOULDBLOCK
```

Type definitions

The platform-specific networking type definitions are:

`os_Socket` specifies the numeric type used to reference an open socket file descriptor.

Example syntax:

```
typedef int os_Socket;
```

Example MPE mapping syntax:

```
typedef os_Socket mpe_Socket;
```

`os_SocketSockLen`

specifies the length of a socket structure.

Example syntax:

```
typedef int os_SocketSockLen;
```

Example MPE mapping syntax:

```
typedef os_SocketSockLen mpe_SocketSockLen;
```

`os_SocketFDSet`
specifies an opaque structure used to define a set of file descriptors.

Example syntax:
`typedef fd_set os_SocketFDSet;`

Example MPE mapping syntax:
`typedef os_SocketFDSet mpe_SocketFDSet;`

`os_SocketLinger`
specifies a toggle. If `l_onoff` is 0, toggling is off. If `l_onoff` is not zero, toggling is on.

Example syntax:
`typedef struct linger os_SocketLinger;`

Example MPE mapping syntax:
`typedef os_SocketLinger mpe_SocketLinger;`

`os_SocketSaFamily`
specifies the socket address family (that is, `MPE_SOCKET_AF_INET4` or `MPE_SOCKET_AF_INET6`).

Example syntax:
`typedef u_char os_SocketSaFamily;`

Example MPE mapping syntax:
`typedef os_SocketSaFamily mpe_SocketSaFamily;`

`os_SocketSockAddr`
specifies the protocol-independent socket address structure.

Example syntax:
`typedef struct sockaddr os_SocketSockAddr;`

Example MPE mapping syntax:
`typedef os_SocketSockAddr mpe_SocketSockAddr;`

`os_SocketIPv4Addr`
holds the IPv4 address.

Example syntax:
`typedef struct in_addr os_SocketIPv4Addr;`

Example MPE mapping syntax:
`typedef os_SocketIPv4Addr mpe_SocketIPv4Addr;`

`os_SocketIPv4McastReq`
 specifies the IPv4 multicast request structure.

Example syntax:

```
typedef struct ip_mreq os_SocketIPv4McastReq;
```

Example MPE mapping syntax:

```
typedef os_SocketIPv4McastReq mpe_SocketIPv4McastReq;
```

`os_SocketIPv4SockAddr`
 specifies the IPv4 socket address structure.

Example syntax:

```
typedef struct sockaddr_in os_SocketIPv4SockAddr;
```

Example MPE mapping syntax:

```
typedef os_SocketIPv4SockAddr mpe_SocketIPv4SockAddr;
```

`os_SocketHostEntry`
 is the host entry returned by calls to
`mpeos_socketGetHostByAddr()` and
`mpeos_socketGetHostByName()`.

Example syntax:

```
typedef ares_hostent os_SocketHostEntry;
```

Example MPE mapping syntax:

```
typedef os_SocketHostEntry mpe_SocketHostEntry;
```

Storage

The file `os_storage.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain the storage primitives.

Definitions

The platform-specific storage primitives definitions are:

`OS_STORAGE_MAX_DISPLAY_NAME_SIZE`
 indicates the maximum size in characters for the display device name.

Example syntax:

```
#define OS_STORAGE_MAX_DISPLAY_NAME_SIZE 40
```

Example MPE mapping syntax:

```
#define MPE_STORAGE_MAX_DISPLAY_NAME_SIZE  
OS_STORAGE_MAX_DISPLAY_NAME_SIZE
```

`OS_STORAGE_MAX_NAME_SIZE`
indicates the maximum size in characters for the storage device name.

Example syntax:

```
#define OS_STORAGE_MAX_NAME_SIZE 63
```

Example MPE mapping syntax:

```
#define MPE_STORAGE_MAX_NAME_SIZE  
OS_STORAGE_MAX_NAME_SIZE
```

`OS_STORAGE_MAX_PATH_SIZE`
indicates the maximum path size.

Example syntax:

```
#define OS_STORAGE_MAX_PATH_SIZE 10
```

Example MPE mapping syntax:

```
#define MPE_STORAGE_MAX_PATH_SIZE  
OS_STORAGE_MAX_PATH_SIZE
```

Type definitions

The platform-specific synchronous primitives type definitions are:

`os_StorageDeviceInfo`

specifies the platform-specific storage device information.

Example syntax:

```
typedef struct _os_StorageDeviceInfo {  
    uint32_t sdmDeviceId;  
    uint32_t dvrDeviceId;  
    Sdm_Volume volumeId[MAX_VOLUMES];  
    char name[OS_STORAGE_MAX_NAME_SIZE + 1];  
    char displayName[  
        OS_STORAGE_MAX_DISPLAY_NAME_SIZE + 1];  
    char systemDataPath[  
        OS_STORAGE_MAX_PATH_SIZE + 1];  
    char appDataPath[OS_STORAGE_MAX_PATH_SIZE + 1];  
    uint16_t status;  
} os_StorageDeviceInfo;
```

Example MPE mapping syntax:

```
typedef os_StorageDeviceInfo mpeos_Storage;
```

Synchronization

The file `os_sync.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain the synchronous primitives.

Definitions

The platform-specific synchronous primitives definitions are:

`none`

| | |
|-------------------------|---|
| Type definitions | The platform-specific synchronous primitives type definitions are: <code>os_Condition</code> resolves the condition identifier type for the MPEOS layer. Example syntax: <code>typedef struct os_Cond_s *os_Condition;</code> Example MPE mapping syntax: <code>typedef os_Condition mpe_Cond;</code> |
| <code>os_Mutex</code> | resolves the mutex identifier type for MPEOS layer. Example syntax: <code>typedef Pk_Mutex* os_Mutex;</code> Example MPE mapping syntax: <code>typedef os_Mutex mpe_Mutex;</code> |
| Thread | The file <code>os_thread.h</code> contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain threads. |
| Definitions | The platform-specific thread definitions are: <code>OS_THREAD_PRIOR_MAX</code> specifies the maximum thread priority. ◆ NOTE: In some operating systems priorities are inverted. Lower value implies higher priority, hence, the MAX priority macro is defined to the minimum value used for application threads. <hr/> Example syntax: <code>#define OS_THREAD_PRIOR_MAX (24)</code> Example MPE mapping syntax: <code>#define MPE_THREAD_PRIOR_MAX OS_THREAD_PRIOR_MAX</code> <code>OS_THREAD_PRIOR_MIN</code> specifies the minimum thread priority. Example syntax: <code>define OS_THREAD_PRIOR_MIN (31)</code> Example MPE mapping syntax: <code>#define MPE_THREAD_PRIOR_MIN OS_THREAD_PRIOR_MIN</code> <code>OS_THREAD_PRIOR_DFLT</code> specifies the default thread priority. Example syntax: <code>#define OS_THREAD_PRIOR_DFLT (28)</code> Example MPE mapping syntax: <code>#define MPE_THREAD_PRIOR_DFLT OS_THREAD_PRIOR_DFLT</code> |

OS_THREAD_PRIOR_INC

specifies the priority setting increment.

Example syntax:

```
#define OS_THREAD_PRIOR_INC (-1)
```

Example MPE mapping syntax:

```
#define MPE_THREAD_PRIOR_INC OS_THREAD_PRIOR_INC
```

OS_THREAD_PRIOR_SYSTEM_HI

specifies a highest (maximum) thread priority for the system implementation threads. OS_THREAD_PRIOR_SYSTEM_HI should be defined to a value higher than the value used for OS_THREAD_PRIOR_SYSTEM_MED.

Example syntax:

```
#define OS_THREAD_PRIOR_SYSTEM_HI  
(OS_THREAD_PRIOR_MAX+1)
```

Example MPE mapping syntax:

```
#define MPE_THREAD_PRIOR_SYSTEM_HI  
OS_THREAD_PRIOR_SYSTEM_HI
```

OS_THREAD_PRIOR_SYSTEM_MED

specifies a medium (moderate) thread priority for the system implementation threads. OS_THREAD_PRIOR_SYSTEM_MED should be defined to a value higher than the value used for OS_THREAD_PRIOR_SYSTEM.

Example syntax:

```
OS_THREAD_PRIOR_SYSTEM_MED (OS_THREAD_PRIOR_MAX+2)
```

Example MPE mapping syntax:

```
#define MPE_THREAD_PRIOR_SYSTEM_MED  
OS_THREAD_PRIOR_SYSTEM_MED
```

OS_THREAD_PRIOR_SYSTEM

specifies the lowest (minimum) thread priority for the system implementation threads. OS_THREAD_PRIOR_SYSTEM should be defined to a value higher than the value used for java.lang.Thread.NORM_PRIORITY.

Example syntax:

```
#define OS_THREAD_PRIOR_SYSTEM(OS_THREAD_PRIOR_MAX+3)
```

Example MPE mapping syntax:

```
#define MPE_THREAD_PRIOR_SYSTEM OS_THREAD_PRIOR_SYSTEM
```

OS_THREAD_STACK_SIZE

specifies the standard thread stack size. The default stack size is large enough for TCP/IP stack usage. This correlates to the default size expected by the virtual machine for threads that get attached to the virtual machine.

Example syntax:

```
#define OS_THREAD_STACK_SIZE (64*1024)
```

Example MPE mapping syntax:

```
#define MPE_THREAD_STACK_SIZE OS_THREAD_STACK_SIZE
```

Type definitions

The platform-specific thread type definitions are:

`os_ThreadStat` specifies the status type for the thread.

Example syntax:

```
typedef uint32_t os_ThreadStat;
```

Example MPE mapping syntax:

```
typedef os_ThreadStat mpe_ThreadStat;
```

`os_ThreadId` specifies the identifier type for the thread.

Example syntax:

```
typedef Pk_Thread *os_ThreadId;
```

Example MPE mapping syntax:

```
typedef os_ThreadId mpe_ThreadId;
```

Time

The file `os_time.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain the system time.

Definitions

The platform-specific time definitions are:

OS_EPOCH_DELTA

specifies the priority setting increment.

Example syntax:

```
#define OS_EPOCH_DELTA (((26 * 365) + 6) * (24 * 60  
* 60))
```

Example MPE mapping syntax:

```
#define MPE_EPOCH_DELTA OS_EPOCH_DELTA
```

Type definitions

The platform-specific time type definitions are:

`os_Time` specifies the operating system time in seconds.

Example syntax:

```
typedef time_t os_Time;
```

Example MPE mapping syntax:

```
typedef os_Time mpe_Time;
```

`os_Clock` specifies a target-specific value usually representing a number of system clock ticks.

Example syntax:

```
typedef clock_t os_Clock;
```

Example MPE mapping syntax:

```
typedef os_Clock mpe_TimeClock
```

`os_Tm` specifies the time as a structure of time components (for example, year, month, day, hour, minutes, seconds).

Example syntax:

```
typedef struct tm os_Tm;
```

Example MPE mapping syntax:

```
typedef os_Tm mpe_TimeTm;
```

`os_TimeVal` specifies a time representation value commonly used in supporting the BSD style MPEOS socket API.

Example syntax:

```
typedef struct timeval os_TimeVal;
```

Example MPE mapping syntax:

```
typedef os_TimeVal mpe_TimeVal;
```

`os_TimeMillis` specifies the time in milliseconds.

Example syntax:

```
typedef long long os_TimeMillis;
```

Example MPE mapping syntax:

```
typedef os_TimeMillis mpe_TimeMillis;
```

Types

The file `os_types.h` contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain the system types.

Definitions

The platform specific, system-types type definition is:

`OS_LIBEXPORT` specifies the types for exporting APIs. The definition is conditional on whether a C or C++ compiler is in use:

Example syntax:

```
#ifdef __cplusplus
#define OS_LIBEXPORT(type, symbol) extern "C" type
symbol
#else
#define OS_LIBEXPORT(type, symbol) extern type
symbol
#endif
```

Example MPE mapping syntax:

```
#define MPE_LIBEXPORT(type, symbol)
OS_LIBEXPORT(type, symbol)
```

Type definitions

The file `os_types.h` contains the following platform specific, integer type definitions:

`typedef signed char os_int8;`
specifies an 8-bit signed integer.

`typedef signed short os_int16;`
specifies a 16-bit signed integer.

`typedef signed long os_int32;`
specifies a 32-bit signed integer.

`typedef long long os_int64;`
specifies a 64-bit signed integer.

`typedef unsigned char os_uint8;`
specifies an 8-bit unsigned integer.

`typedef unsigned short os_uint16;`
specifies a 16-bit unsigned integer.

`typedef unsigned long os_uint32;`
specifies a 32-bit unsigned integer.

`typedef unsigned long long os_uint64;`
specifies a 64-bit unsigned integer.

Utilities

The file os_util.h contains mandatory definitions, data types and structures that bind to platform-specific values, and internal implementation structures used to maintain utilities.

Definitions

The platform-specific utilities definitions are:

none

Type definitions

The platform-specific utilities type definitions are:

os_JmpBuf specifies a setjmp/longjmp implementation buffer.

Example syntax:

```
typedef jmp_buf os_JmpBuf;
```

Example MPE mapping syntax:

```
typedef os_JmpBuf mpe_JmpBuf;
```

Host-device applications

This section describes what host-device applications are, how they can be supported and provisioned within the OCAP stack, and options available to the porting engineer regarding their design and organization.

Overview

Host-device applications are applications that use or provide functionality of a set-top platform that is outside the scope of the functionality provided by the OCAP APIs. This extended functionality can be support for legacy features of the platform not addressed by OCAP or it can be support for newer technologies not yet addressed by OCAP.

The most common host-device applications are legacy applications that were provided by manufactures prior to OCAP. Such legacy applications may continue to add value by supporting valuable features of the platform that are not provided by the OCAP APIs. For example, a host-device application might provide the ability to configure and use alternate analog or digital input sources, select alternate audio output formats, or configure hardware devices outside of the scope of OCAP.

OCAP does make provisions within the specification for the existence and use of host-device applications within the realm of an OCAP device. The specification requires that these applications, when using or competing for use of resources managed under OCAP, must have Xlet application front-ends so the OCAP rules governing resource allocation between Xlet applications can be applied and administered. This allows host-device applications to be launched and controlled in the same manner as other Xlets, thus providing a uniform deployment methodology for all applications.

In addition to being wrapped with an Xlet front-end, a host-device application that in any way interacts with functionality addressed by OCAP needs to behave as a “well behaved” Xlet. Simply applying an Xlet interface to a host-device application that in some way takes control of any of the features addressed by OCAP could be considered non-compliant and “ill behaved.”

Provisioning

A host application with the appropriate Xlet interface can optionally be built directly into the stack for convenience.

The preferred method of provisioning host-device applications into the OCAP stack is to first use the design options described in the *Organization and design* section to supply the APIs and associated Xlet class files. Once the code base for the application is fully integrated, the applications can be made available for execution at runtime by adding an application description to the `hostapps.properties` file that gets statically built into the system. The `hostapps.properties` file is used to statically describe host-device applications in a manner very similar to the way XAITs and AITs are used to signal applications dynamically. It specifies the application identifier, the application control code (for example, `AUTOSTART`, `PRESENT`), the application name, the base or root directory, the initial Xlet class file, the applications priority, an optional class path extension, an optional abstract service identifier, and any execution parameters to be passed to the application when the application is started. The intent of the `hostapps.properties` file is to provide a uniform method of provisioning host-device applications in a way that is analogous to the way non-host-device applications are provisioned to OCAP devices.

Organization and design

Depending on the nature of the feature and whether that feature involves some resource managed by the OCAP specification (for example, graphics device), it may be a fairly simple task for the manufacturer to either completely convert the legacy application to an Xlet or simply to provide the required Xlet front-end. Set-top box manufacturers that want to provide host-device applications that have Xlet interfaces for negotiating OCAP resources have a couple of design options available.

One option would be to have a self-contained Xlet application and its Java JNI access methods use a JNI native library as a separate component module to gain access to the native manufacturer support. For various functional and logistic reasons this brute-force approach is not recommended and should be avoided, and therefore, the specifics of this approach will not be discussed.

The preferred method of supplying the Java and native manufacturer support is to extend the base OCAP stack. This can be done by bundling the Java APIs that grant access to the native functionality as one or more separate component jar files and bundling the additional JNI native library as a separate component extension module. The additional extension jar(s) must be listed in both the class path list and in the API package(s) list listed in the `OCAP.extensions` list defined in the target's `ini` file. For example, the `ini` file entry below demonstrates the addition of a manufacture API package of extensions:

```
OCAP.extensions=com.abc.extensions
```

This design is the most flexible means of extending the OCAP stack to include manufacturer support. It provides a much greater degree of separation of the Xlet application, the general OCAP stack proper, and the additional manufacturer code. One of the main advantages to this strategy is that native extension code can be built without requiring access to the entire OCAP stack source and build environment. A separate smaller scale build environment can be constructed to provide the ability to create the manufacturer's custom extension module.

Similarly, the Java code can be constructed as a separate set of one or more Java packages. By adding these packages to the class path list, they are included in the set of classes allowed to be loaded by the system class loader. In addition, including them in the `OCAP.extensions` property defined in the `ini` file allows the application class loader to defer searching for these classes contained in those packages to the system class loader, thus allowing them to be found for all referencing host-device applications. In order to cause the JNI native library to be loaded in support of these APIs, one of these implementation classes must perform the `System.loadLibrary()` function to load the native library. Ideally, the first API class referenced by a client host-device application will contain this call within a static initializer.

If a manufacturer wishes to restrict general use of the extension APIs and limit accessibility to only their own set of manufacturer host-device applications, the APIs can be written to enforce applications to be associated with either particular or a general set of permissions. To verify that the calling application has the appropriate permissions, the APIs would need to perform a security check similar to the following:

```
SecurityManager sm = System.getSecurityManager();
if ( sm != null )
    sm.checkPermission( ManufacturerABCPermissionXYZ );
```

Specific manufacturer-defined permissions can be assigned to signed host-device applications using the `OCAP.security.policy.<i>` property that can be defined in the `ini` file. For example:

```
OCAP.security.policy.0=com.abc.apps.HostAppPermissionXYZ
```

As an alternative, a more general and wide-open permissions approach can be used to limit access to only super host-device applications. In this case the manufacturer host-device applications would be signaled with application identifiers in the "super host-device application" range, which would cause the application manager to automatically assign the set of `AllPermission` permissions to these applications. Since these are the only applications assigned this set of permissions, they are the only applications passing the security check that would need to be within the manufacturer APIs.

Third-party software

Operating systems may not contain all the OCAP functionality requirements. Following are some packages Vidiom uses to create missing functionality for other ports:

ARES

Asynchronous Resolver (ARES) is used if an operating system lacks the functionality to provide a name resolver. With some operating systems, ARES is part of the implementation of the MPEOS networking API (for example, `mpe_socketGetHostByName()`). ARES is intended for applications that need to perform Domain Name Server (DNS) queries without blocking, or need to perform multiple DNS queries in parallel. The primary examples of such applications are servers that communicate with multiple clients and programs with graphical user interfaces.



For more information about the asynchronous resolver library, refer to the [README file located in the %OCAPROOT%\other\ares-1.1.1 directory](#).

DirectFB

is used if an operating system lacks the functionality to directly support what is needed for OCAP graphics.



For more information about DirectFB, refer to the [README_VIDIOM.txt file located in the %OCAPROOT%\other\DirectFB directory](#).

DirectFB provides developers with hardware graphics acceleration, input device handling and abstraction, integrated windowing system with support for translucent windows, and multiple display layers. DirectFB is a complete hardware abstraction layer with software fallbacks for every graphics operation that is not supported by the underlying hardware.

FreeType2

is used to support Portable Font Resource (PFR) fonts. The FreeType2 software font



For more information about FreeType2, refer to the [%OCAPROOT%\other\FreeType2\docs directory](#).

engine, is designed to be small, efficient, easy to customize, and portable while able to produce high-quality output (glyph images). It can be used in graphics libraries, display servers, font conversion tools, text image generation tools, and many other products.

Launching the JVM

The initial instantiation of the JVM is accomplished through the JNI interface. Currently, the OCAP stack assumes that all JVM implementations will provide the JNI interface and, therefore, the JVM creation and instantiation code is located within the platform-independent MPE layer. The JVM instantiation function is called `mpe_jvmExecuteMain()` and is called from the MPEOS porting layer (`mpeos_main.c`). The instantiation process is typically performed on the initial thread of execution, but can be done on any thread as long as the thread is configured adequately for the particular JVM in use. For example, the Esmertec JVM requires that all threads use a 64K stack image.

Subsequent to instantiating the JVM, the `mpe_jvmExecuteMain()` API also launches the initial OCAP class. This initial class causes additional threads to be created in support of the entire OCAP stack. This initial class invocation is essentially an asynchronous process. This means that the implementation of `mpeos_main.c` must provide a means to handle the initial thread once it returns from the initial class invocation. On some systems, it may be possible to simply allow this initial thread to terminate. On other systems, it may be required that this thread continue to execute indefinitely, in which case the port could either suspend the thread or possibly make additional use of this initial thread (for example, for event processing). The strategy for how to most efficiently coordinate the management of the initial thread is entirely platform-specific and part of the overall responsibility of the implementation of the support code within `mpeos_main.c`.

6 MPEOS Testing

Overview

This chapter addresses the strategy and procedure for unit testing of the Multimedia Platform Environment Operating System (MPEOS) Application Programming Interfaces (API) during the early phases of the porting process.

The strategy for the MPEOS API unit testing framework is to allow for incremental testing of the MPEOS APIs as they are being developed. By allowing for a targeted, incremental testing approach, the porting process is easily divided into discrete coding and verification phases. This allows any base-level APIs, which more complex APIs may rely on for their implementation, to be developed and verified first. For example, typically the basic-operating system APIs (memory allocation and threads) would be developed and verified before more complex services (media), meaning the media-based APIs could use in their implementation the basic operating system-level MPEOS APIs.

Before reading this chapter, you should be:

- ◆ familiar with how to generate the OCAP stack
- ◆ familiar with how to flash the OCAP stack image onto the host device

After reading this chapter, you should:

- ◆ be familiar with the testing process
- ◆ know how to read the output from the testing harness

This incremental development strategy is in part supported by how the MPEOS APIs are built. As previously mentioned, the MPEOS APIs are built into a separate library. By compiling the APIs into a stand-alone library, the testing support code can be selective and reference (include) only the APIs available at any given time during the porting process. Therefore, a simplified view of the incremental approach would consist of the following basic steps:

1. Code and compile a subset of the MPEOS APIs into the MPEOS API library (for example, `libmpeos.a`).
2. Modify the main testing code to reference just the targeted tests of the subset of MPEOS APIs to be verified.
3. Build the subset of tests into a native test program that links against the MPEOS API library to resolve the `mpeos_` API references.
4. Test and debug the target MPEOS API subset.



For detailed information on modifying the test code, refer to *Modifying the test code* section.

In enabling this incremental approach, the MPEOS API testing support has been divided into a separate sub-directory location within the overall MPE unit testing framework. The MPEOS API testing framework for a given target should be located in the following directory:

`%OCAPROOT%\mpe\test\os\target operating system`

This directory should contain all of the sources and build files necessary to create a stand-alone test program that you can execute and debug on the target platform.

Most targets will only contain a few files within this sub-directory. The main file components and their roles within the MPEOS API testing framework are as follows:

`main.c` is the base source file containing the functional code requirements for the MPEOS API unit tests to execute as an application on the target platform. In the most simplistic case, this is just a standard C, `main(int argc, char **argv)` type entry point. This file will contain whatever code needs to be in place to interface with the native operating system to support a stand-alone application. To invoke the actual testing framework, the main code, after performing any target-specific application initialization, needs to call the function `TestRunner()`. This is the main function invoked to perform specific MPEOS API testing.

`Makefile` is the actual makefile used to specify the build rules of the target platform. For most platforms, it is a makefile, but it would not necessarily have to be a makefile. For example, on Windows it could be a project file. In any case, it is whatever the target platform build tools require and is specified within the `build.xml` file as the mechanism to construct the test application target.

`TestRunner.c` is the source file containing the main entry point testing function. `TestRunner()` calls each of the specific test routines under scrutiny. Each category of MPEOS API services within the MPEOS API library has its own set of test routines and each category set is further divided into functional subsets. Therefore, with available wrapper functions, you can invoke testing of:

- ◆ the entire category
- ◆ specific functional subsets
- ◆ a specific list of tests within a category subset

Another important function located within this file is the `vte_agent_Log()`, which is responsible for displaying tests result logs.

Modifying the test code

You can modify the test code to select a single test, a group of related tests, or all the tests. The modification is done directly to the `TestRunner.c` file located in the directory:

`%OCAPROOT%\mpe\test\os\target operating system`

Comment out the test(s) you do not want to run and recompile the testing application for the target. The test functions currently available for MPEOS APIs include:

- ◆ *All*
- ◆ *Group*
- ◆ *Individual*

All

To test all the MPEOS APIs use the following function:

```
test_sysRunAllTests()
    executes all of the system tests.
```

Group

Group tests include the following functions:

```
test_sysRunDbgTests()
    execute the debug system tests.

test_sysRunEventTests()
    execute the event system tests.

test_sysRunMathTests()
    execute floating-point math tests (IEEE-754 math
    conformance).

test_sysRunMemTests()
    executes the memory system tests.

test_sysRunThreadTests()
    executes the thread system tests.

test_sysRunTimeTests()
    execute the time system tests.

test_sysRunStressTests()
    execute several system tests including thread operation,
    synchronization, events, and memory tests.

test_sysRunSyncTests()
    execute the synchronous system tests.

test_sysRunUtilTests()
    execute the miscellaneous utility system tests.
```

Individual

Individual test are used to validate the individual functions. The test have been divided into the following categories based on functionality:

| | |
|-----------------|-------------|
| Event | Front panel |
| Memory | Network |
| Synchronization | Time |

Event

These individual tests are used to validate the Event API:

- `test_sysRunEventPingPongTest()`
verifies multiple event synchronization between two threads.
- `test_sysRunEventQueueNewTest()`
verifies `mpeos_eventQueueNew()` creates a new event queue.
- `test_sysRunEventQueueNextTest()`
verifies `mpeos_eventQueueNext()` gets the next event from the specified queue (non-blocking).
- `test_sysRunEventQueueSendTest()`
verifies `mpeos_eventQueueSend()` posts the specified event to the specified queue.

Front panel

These individual tests are used to validate the Front Panel API:

- `test_sysRunFpTests()`
verifies MPEOS, front-panel functionality.

Memory

These individual tests are used to validate the Memory API:

- `test_sysRunMemAllocH()`
verifies `mpeos_memAllocH()` allocates a block of memory via a handle.
- `test_sysRunMemAllocP()`
verifies `mpeos_memAllocP()` allocates a block of memory via a pointer.
- `test_sysRunMemFreeP()`
verifies `mpeos_memFreeP()` frees a block of memory via a pointer.
- `test_sysRunMemGetSizeTest()`
verifies `mpeos_memGetSize()` gets a specified size of system memory.
- `test_sysRunMemGetLargestFree()`
verifies `mpeos_memGetLargestFree()` gets the largest block of free system memory.
- `test_sysRunMemLargestFreeTest()`
verifies `mpeos_memGetLargestFree` gets the largest block of free system memory.
- `test_sysRunMemReallocP()`
verifies `mpeos_memReallocP()` changes the size of the memory block via a pointer.

Network

These individual tests are used to validate the Network API:

```
test_netRunAllTests()
    verifies all MPEOS network tests.

test_netRunLookupTests()
    verifies MPEOS, network-lookup tests.

test_netRunConnectTests()
    verifies the MPEOS network connection process.

test_netRunOptionsTests()
    verifies the MPEOS network options functionality.

test_netRunSelectTests()
    verifies the MPEOS network select process.

test_mpe_netReadWriteTests()
    verifies the MPEOS network read/write functionality.
```

Synchronization

These individual tests are used to validate the Synchronization API:

```
test_sysRunSyncCondDeleteTest()
    verifies mpeos_condDelete() deletes a condition object.

test_sysRunSyncCondGetSetTest()
    verifies mpeos_condGet() gets exclusive access to a
    condition object and mpeos_condSet() sets the specified
    condition variable to TRUE.

test_sysRunSyncCondNewTest()
    verifies mpeos_condNew() creates a new condition
    synchronization object.

test_sysRunSyncCondPingPongTest()
    verifies mpeos_condSet() sets the specified condition object
    to TRUE and mpeos_condGet() gets exclusive access of the
    condition object.

test_sysRunSyncCondWaitForTest()
    verifies mpeos_condWaitFor() waits and gets a specified
    amount of time for a condition variable.

test_sysRunSyncMutexAcquireTest()
    verifies mpeos_mutexAcquire() acquires ownership of the
    target mutex.

test_sysRunSyncMutexAcquierTryTest()
    verifies mpeos_mutexAcquireTry() non-blocking version of
    get/acquire/lock.

test_sysRunSyncMutexNewTest()
    verifies mpeos_mutexNew() creates a new mutex
    variable/object/structure.
```

Time

These individual tests are used to validate the Time API:

```
test_sysRunTimeClockTest()
    verifies mpeos_timeClock() gets the amount of CPU time
    used by the current process or thread.

test_sysRunTimeClockToMillisTest()
    verifies mpeos_timeClockToMillis() converts clock time to
    milliseconds.

test_sysRunTimeClockToTimeTest()
    verifies mpeos_timeClockToTime() converts an
    mpe_TimeClock value to an mpe_Time value.

test_sysRunTimeClockTicksTest()
    verifies mpeos_timeClockTicks() gets the number of clock
    ticks per second.

test_sysRunTimeGetTest()
    verifies mpeos_timeClock() gets the current time.

test_sysRunTimeMillisToClockTest()
    verifies mpeos_timeMillisToClock() converts a time value
    to system clock ticks.

test_sysRunTimeTmToTimeTest()
    verifies mpeos_timeTmToTime() converts an mpe_TimeTm
    structure to an mpe_Time value.

test_sysRunTimeToDateTest()
    verifies mpeos_timeToDate() converts the time value to the
    local date value.
```

Understanding the testing framework

The actual test cases themselves have been implemented with an open, public domain testing framework called CUTest, which was simplistically modelled after the Java JUnit framework often used for unit testing of Java classes. The main difference being the individual test suites must be explicitly constructed and invoked within the testing code with the CUTest framework, versus the class reflection method used with the JUnit framework. Therefore, the category and category subset test cases invoked from TestRunner() are simply helper functions that construct specific CUTest test suites of individual test cases for the MPEOS APIs.

Like the Java JUnit framework, the CUTest framework uses a set of assertion statements allowing the test cases to assert specific results and conditions associated with a particular MPEOS API. The assertion statements test for the specified conditions and on failure will log an associated text string with file name and line number reflecting the location of the assertion failure. When the test run of the individual test suites recompile, the CUTest framework reports the assertion statement text string logs along with the number of successful tests completed. Using the assertion logs and the CUTest case sources, you will be able to identify the specific test cases that are failing. Therefore, during the testing process, you will need to consult the actual test cases to gain a better understanding of the specifics of the failing test(s). The categorized test cases are located within the following source files:

```
mpe\test\mpe\mpe manager specific category\category subset.c
```

Where the *mpe manager specific category* represents a sub-directory for a functional category (for example, base operating system [sysmgr], or graphics [dispmgr], etc.) and *category subset.c* contains the subset of test suites within the category (for example, *test_sys_thread.c*, *test_sys_mem.c*, etc.).

The CUTest framework is setup to report the various test result logs. To do this, make a callback to a function named *vte_agent_Log(const char * format, ...)*, which must be implemented as part of the porting process. The *vte_agent_Log()* signature matches that of the standard C-library *printf()* function. Typically, it simply passes the log string to the *vprintf* function available in most standard C-library implementations. Implement *vte_agent_Log()* with whatever is necessary for the target platform to make the text logs available for review.



For detailed information on implementation of the CUTest framework, consult the implementation files (*catest.c* and *catest.h*) found in the *mpe\test\common* directory.

Currently, unlike other portions of the OCAP stack, the MPEOS API testing support can only be built from the target specific location:

`%OCAPROOT%\mpe\test\os\itarget operating system`

Because the framework for testing the MPEOS APIs shares the test code with the higher-level MPE API testing, the sources must be built with different compiler options.

Target device requirements for testing

The target device must supply the framework for:

- ◆ executing the base source file containing the functional code (`main.c`)
- ◆ the logging framework to allow the results to be viewed

CAP Closed-Captioning API

Overview

The closed-captioning Application Programming Interface (API) is used by the OpenCable Application Platform (OCAP) stack to provide support for displaying captions encoded in analog and digital video streams.

Before reading this chapter, you should be familiar with:

- ◆ what closed captioning is and how it works
- ◆ the closed-captioning OCAP requirements, including Electronics Industry Association (EIA)-708B and EIA-608B standards for decoding closed-caption data
- ◆ the registry of constants in the OCAP specification

After reading this chapter, you should be able to port the OCAP required functions for:

- ◆ turning on and off closed captioning
- ◆ selecting closed-captioning services
- ◆ querying closed-captioning capabilities
- ◆ setting closed-captioning attributes

Definitions

The following definitions, which are defined in `mpeos_caption.h`, are used by the closed-captioning functions:

`MPE_CC_COLOR`

`MPE_CC_EMBEDDED_COLOR`

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

`MPE_CC_COLOR`

`MPE_CC_COLOR` is a utility macro that converts RGB888 (8-bits per component) color values already reduced to 2 bits into a closed-caption color defined as RGB222 (2-bits per component). This macro is not required for porting the closed captioning to a different platform.

`MPE_CC_COLOR` is defined as follows:

```
#define MPE_CC_COLOR(r,g,b) ( ((r)<<4) | ((g)<<2) | (b) )
```

`MPE_CC_EMBEDDED_COLOR`

`MPE_CC_EMBEDDED_COLOR` indicates an embedded value is used for the color attributes of the closed-captioning foreground, background, and border.

`MPE_CC_EMBEDDED_COLOR` is an optional value for `mpe_CcColor`.

`MPE_CC_EMBEDDED_COLOR` is defined as follows:

```
#define MPE_CC_EMBEDDED_COLOR (0xff000000)
```

Data types and structures

The following data types and structures, which are defined in `mpeos_caption.h`, `mpe_error.h`, and `mpe_types.h`, are used by the closed-captioning functions:

| | |
|--------------------------------------|------------------------------------|
| <code>mpe_CcAnalogServiceMap</code> | <code>mpe_CcAnalogServices</code> |
| <code>mpe_CcAttribType</code> | <code>mpe_CcAttributes</code> |
| <code>mpe_CcBorderType</code> | <code>mpe_CcColor</code> |
| <code>mpe_CcDigitalServiceMap</code> | <code>mpe_CcDigitalServices</code> |
| <code>mpe_CcError</code> | <code>mpe_CcFontSize</code> |
| <code>mpe_CcFontStyle</code> | <code>mpe_CcOpacity</code> |
| <code>mpe_CcState</code> | <code>mpe_CcTextStyle</code> |
| <code>mpe_CcType</code> | <code>mpe_Error</code> |

`mpe_CcAnalogServiceMap`

`mpe_CcAnalogServiceMap` specifies an analog service map.
`mpe_CcAnalogServiceMap` is defined in `mpeos_caption.h` as follows:

```
typedef uint8_t mpe_CcAnalogServiceMap;
```

`mpe_CcAnalogServices`

`mpe_CcAnalogServices` specifies an analog closed-captioning services.
`mpe_CcAnalogServices` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcAnalogServices {
    MPE_CC_ANALOG_SERVICE_NONE = 0,
    MPE_CC_ANALOG_SERVICE_CC1 = 1000,
    MPE_CC_ANALOG_SERVICE_CC2 = 1001,
    MPE_CC_ANALOG_SERVICE_CC3 = 1002,
    MPE_CC_ANALOG_SERVICE_CC4 = 1003,
    MPE_CC_ANALOG_SERVICE_T1 = 1004,
    MPE_CC_ANALOG_SERVICE_T2 = 1005,
    MPE_CC_ANALOG_SERVICE_T3 = 1006,
    MPE_CC_ANALOG_SERVICE_T4 = 1007
} mpe_CcAnalogServices;
```

where

`MPE_CC_ANALOG_SERVICE_NONE`
 specifies no analog service.

`MPE_CC_ANALOG_SERVICE_CC1`
 specifies the primary synchronous caption service that is the primary language captioning data that should be in sync with the sound, preferably to a specific frame.

`MPE_CC_ANALOG_SERVICE_CC2`
 specifies the special non-synchronous channel that carries data that may be intended to augment information carried in the program. It does not need to be in sync with the sound.

MPE_CC_ANALOG_SERVICE_CC3

specifies the secondary synchronous caption service and is an alternate captioning data channel usually used for second language captions.

MPE_CC_ANALOG_SERVICE_CC4

specifies the secondary special non-synchronous channel.

MPE_CC_ANALOG_SERVICE_T1

specifies the first text service.

MPE_CC_ANALOG_SERVICE_T2

specifies the second text service and may be used for the transmission of broadcast related URLs.

MPE_CC_ANALOG_SERVICE_T3

specifies the third text service and should be used only if MPE_CC_ANALOG_SERVICE_T1 and MPE_CC_ANALOG_SERVICE_T2 are not sufficient.

MPE_CC_ANALOG_SERVICE_T4

specifies the fourth text service and should be used only if MPE_CC_ANALOG_SERVICE_T1, MPE_CC_ANALOG_SERVICE_T2, and MPE_CC_ANALOG_SERVICE_T3 are not sufficient.

mpe_CcAttribType

mpe_CcAttribType specifies the type of attribute to set. For example, MPE_CC_ATTRIB_FONT_COLOR indicates the application is setting the closed-caption text color. *mpe_CcAttribType* is defined in *mpeos_caption.h* as follows:

```
typedef enum mpe_CcAttribType {
    MPE_CC_ATTRIB_FONT_COLOR = 0x0001,
    MPE_CC_ATTRIB_BACKGROUND_COLOR = 0x0002,
    MPE_CC_ATTRIB_FONT_OPACITY = 0x0004,
    MPE_CC_ATTRIB_BACKGROUND_OPACITY = 0x0008,
    MPE_CC_ATTRIB_FONT_STYLE = 0x0010,
    MPE_CC_ATTRIB_FONT_SIZE = 0x0020,
    MPE_CC_ATTRIB_FONT_ITALIC = 0x0040,
    MPE_CC_ATTRIB_FONT_UNDERLINE = 0x0080,
    MPE_CC_ATTRIB_BORDER_TYPE = 0x0100,
    MPE_CC_ATTRIB_BORDER_COLOR = 0x0200,
    MPE_CC_ATTRIB_WIN_COLOR = 0x0400,
    MPE_CC_ATTRIB_WIN_OPACITY = 0x0800,
    MPE_CC_ATTRIB_MAX
} mpe_CcAttribType;
```

where

MPE_CC_ATTRIB_FONT_COLOR

specifies the color for the caption characters.

MPE_CC_ATTRIB_BACKGROUND_COLOR

specifies the background color for the caption.

MPE_CC_ATTRIB_FONT_OPACITY

specifies the opacity for the caption characters.

MPE_CC_ATTRIB_BACKGROUND_OPACITY
 specifies a background opacity for the caption.

MPE_CC_ATTRIB_FONT_STYLE
 specifies a font style.

MPE_CC_ATTRIB_FONT_SIZE
 specifies the point size of the font.

MPE_CC_ATTRIB_FONT_ITALIC
 specifies an italic font.

MPE_CC_ATTRIB_FONT_UNDERLINE
 specifies an underlined font.

MPE_CC_ATTRIB_BORDER_TYPE
 specifies a border type.

MPE_CC_ATTRIB_BORDER_COLOR
 specifies a colored border.

MPE_CC_ATTRIB_WIN_COLOR
 specifies a colored window.

MPE_CC_ATTRIB_WIN_OPACITY
 specifies an opaque window.

MPE_CC_ATTRIB_MAX
 specifies the maximum value that can be stored in `mpe_CcAttribType`.

`mpe_CcAttributes`

`mpe_CcAttributes` stores the value of the closed-captioning attributes. If the attribute is not set, by default, the attribute value defined in the Moving Picture Experts Group (MPEG) stream is used. An attribute can also be set to a specific value. For example, if `MPE_CC_ATTRIB_FONT_COLOR` is set, the value (representing a color) is stored in `charFgColor`. An attribute can be set to its embedded value by using a specific value. For example, `MPE_CC_OPACITY_EMBEDDED` resets the `OPACITY` value to the one embedded in the closed-captioning stream. `mpe_CcAttributes` is defined in `mpeos_caption.h` as follows:

```
typedef struct mpe_CcAttributes {
    mpe_CcColor charBgColor;
    mpe_CcColor charFgColor;
    mpe_CcColor winColor;
    mpe_CcOpacity charBgOpacity;
    mpe_CcOpacity charFgOpacity;
    mpe_CcOpacity winOpacity;
    mpe_CcFontSize fontSize;
    mpe_CcFontStyle fontStyle;
    mpe_CcTextStyle fontItalic;
    mpe_CcTextStyle fontUnderline;
    mpe_CcBorderType borderType;
    mpe_CcColor borderColor;
} mpe_CcAttributes;
```

where

| | |
|----------------------------|---|
| <code>charBgColor</code> | specifies the background color for the caption. <code>mpe_CcColor</code> is defined in the <i>mpe_CcColor</i> section. Additionally, <code>MPE_CC_EMBEDDED_COLOR</code> is an optional value for <code>mpe_CcColor</code> . |
| <code>charFgColor</code> | specifies the color for the caption characters. <code>mpe_CcColor</code> is defined in the <i>mpe_CcColor</i> section. Additionally, <code>MPE_CC_EMBEDDED_COLOR</code> is an optional value for <code>mpe_CcColor</code> . |
| <code>winColor</code> | specifies the window color. <code>mpe_CcColor</code> is defined in the <i>mpe_CcColor</i> section. Additionally, <code>MPE_CC_EMBEDDED_COLOR</code> is an optional value for <code>mpe_CcColor</code> . |
| <code>charBgOpacity</code> | specifies the opacity of the background color for the caption. <code>mpe_CcOpacity</code> is defined in the <i>mpe_CcOpacity</i> section. |
| <code>charFgOpacity</code> | specifies the opacity for the caption characters. <code>mpe_CcOpacity</code> is defined in the <i>mpe_CcOpacity</i> section. |
| <code>winOpacity</code> | specifies the window opacity. <code>mpe_CcOpacity</code> is defined in the <i>mpe_CcOpacity</i> section. |
| <code>fontSize</code> | specifies the point size of the font. <code>mpe_CcFontSize</code> is defined in the <i>mpe_CcFontSize</i> section. |
| <code>fontStyle</code> | specifies the font style. <code>mpe_CcFontStyle</code> is defined in the <i>mpe_CcFontStyle</i> section. |
| <code>fontItalic</code> | specifies an italic font. <code>mpe_CcFontStyle</code> is defined in the <i>mpe_CcFontStyle</i> section. |
| <code>fontUnderline</code> | specifies an underlined font. <code>mpe_CcFontStyle</code> is defined in the <i>mpe_CcFontStyle</i> section. |
| <code>borderType</code> | specifies the window's border type. <code>mpe_CcBorderType</code> is defined in the <i>mpe_CcBorderType</i> section. |
| <code>borderColor</code> | specifies the window's border color. <code>mpe_CcColor</code> is defined in the <i>mpe_CcColor</i> section. Additionally, <code>MPE_CC_EMBEDDED_COLOR</code> is an optional value for <code>mpe_CcColor</code> . |

`mpe_CcBorderType`

`mpe_CcBorderType` specifies the type of border on the closed-captioning window. `mpe_CcBorderType` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcBorderType {
    MPE_CC_BORDER_TYPE_EMBEDDED = -1,
    MPE_CC_BORDER_TYPE_NONE,
    MPE_CC_BORDER_TYPE_RAISED,
    MPE_CC_BORDER_TYPE_DEPRESSED,
```

```

        MPE_CC_BORDER_TYPE_UNIFORM,
        MPE_CC_BORDER_TYPE_SHADOW_LEFT,
        MPE_CC_BORDER_TYPE_SHADOW_RIGHT,
        MPE_CC_BORDER_TYPE_MAX
    } mpe_CcBorderType;

```

where

MPE_CC_BORDER_TYPE_EMBEDDED
 indicates the embedded border.

MPE_CC_BORDER_TYPE_NONE
 indicates no border.

MPE_CC_BORDER_TYPE_RAISED
 indicates a raised border.

MPE_CC_BORDER_TYPE_DEPRESSED
 indicates a depressed border.

MPE_CC_BORDER_TYPE_UNIFORM
 indicates a flat border.

MPE_CC_BORDER_TYPE_SHADOW_LEFT
 indicates a left shadow on the border.

MPE_CC_BORDER_TYPE_SHADOW_RIGHT
 indicates a right shadow on the border.

MPE_CC_BORDER_TYPE_MAX
 specifies the maximum value that can be stored in
 mpe_CcBorderType.

mpe_CcColor

mpe_CcColor specifies a closed-captioning Red, Green, and Blue (RGB) values as defined by the EIA708B standard for digital closed captioning. mpe_CcColor defines the format as an RGB222 (2 bits for red, green, and blue components). Before the operating system function is called to set the color, MPE_CC_COLOR is used to parse this 32-bit value. mpe_CcColor is defined in mpeos_caption.h as follows:

```
typedef uint32_t mpe_CcColor;
```

mpe_CcDigitalServiceMap

mpe_CcDigitalServiceMap specifies a digital service map.
 mpe_CcDigitalServiceMap is defined in mpeos_caption.h as follows:

```
typedef uint64_t mpe_CcDigitalServiceMap;
```

mpe_CcDigitalServices

`mpe_CcDigitalServices` specifies a digital closed captioning services.
`mpe_CcDigitalServices` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcDigitalServices {
    MPE_CC_DIGITAL_SERVICE_NONE = 0
} mpe_CcDigitalServices;
```

where

`MPE_CC_DIGITAL_SERVICE_NONE`
 specifies there are no digital services available.

mpe_CcError

`mpe_CcError` specifies the error codes for closed captioning. `mpe_CcError` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcError {
    MPE_CC_ERROR_NONE = MPE_SUCCESS,
    MPE_CC_ERROR_INVALID_PARAM = MPE_EINVAL,
    MPE_CC_ERROR_OS
} mpe_CcError;
```

where

`MPE_CC_ERROR_NONE`
 indicates the successful completion of an `mpe_` or `mpeos_` function call.

`MPE_CC_ERROR_INVALID_PARAM`
 indicates at least one input parameter passed to the MPE function was invalid.

`MPE_CC_ERROR_OS`
 indicates an operating system error occurred.

mpe_CcFontSize

`mpe_CcFontSize` specifies the point size of the font for the closed captioning. `mpe_CcFontSize` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcFontSize {
    MPE_CC_FONT_SIZE_EMBEDDED = -1,
    MPE_CC_FONT_SIZE_SMALL,
    MPE_CC_FONT_SIZE_STANDARD,
    MPE_CC_FONT_SIZE_LARGE,
    MPE_CC_FONT_SIZE_MAX
} mpe_CcFontSize;
```

where

`MPE_CC_FONT_SIZE_EMBEDDED`
 indicates an embedded font size.

`MPE_CC_FONT_SMALL`
 indicates the small size font.

`MPE_CC_FONT_SIZE_STANDARD`
 indicates the standard size font.

MPE_CC_FONT_SIZE_LARGE
indicates the large size font.

MPE_CC_FONT_SIZE_MAX
specifies the maximum value that can be stored in
`mpe_CcFontSize`.

mpe_CcFontStyle

`mpe_CcFontStyle` specifies the style of the font for the closed captioning.
`mpe_CcFontStyle` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcFontStyle {
    MPE_CC_FONT_STYLE_EMBEDDED = -1,
    MPE_CC_FONT_STYLE_DEFAULT,
    MPE_CC_FONT_STYLE_MONOSPACED_SERIF,
    MPE_CC_FONT_STYLE_PROPORTIONAL_SERIF,
    MPE_CC_FONT_STYLE_MONOSPACED_SANSERIF,
    MPE_CC_FONT_STYLE_PROPORTIONAL_SANSERIF,
    MPE_CC_FONT_STYLE_CASUAL,
    MPE_CC_FONT_STYLE_CURSIVE,
    MPE_CC_FONT_STYLE_SMALL_CAPITALS,
    MPE_CC_FONT_STYLE_MAX
} mpe_CcFontStyle;
```

where

MPE_CC_FONT_STYLE_EMBEDDED
indicates an embedded style is used.

MPE_CC_FONT_STYLE_DEFAULT
indicates the default font.

MPE_CC_FONT_STYLE_MONOSPACED_SERIF
indicates the monospacing serif font.

MPE_CC_FONT_STYLE_PROPORTIONAL_SERIF
indicates the proportional serif font.

MPE_CC_FONT_STYLE_MONOSPACED_SANSERIF
indicates the monospacing sans serif font.

MPE_CC_FONT_STYLE_PROPORTIONAL_SANSERIF
indicates the proportional sans serif font.

MPE_CC_FONT_STYLE_CASUAL
indicates the casual font.

MPE_CC_FONT_STYLE_CURSIVE
indicates the cursive font.

MPE_CC_FONT_STYLE_SMALL_CAPITALS
indicates the small capital font.

MPE_CC_FONT_STYLE_MAX
specifies the maximum value that can be stored in
`mpe_CcFontStyle`.

mpe_CcOpacity

`mpe_CcOpacity` specifies the opaque level of the closed captioning.
`mpe_CcOpacity` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcOpacity {
    MPE_CC_OPACITY_EMBEDDED = -1,
    MPE_CC_OPACITY_TRANSPARENT,
    MPE_CC_OPACITY_TRANSLUCENT,
    MPE_CC_OPACITY_SOLID,
    MPE_CC_OPACITY_FLASHING,
    MPE_CC_OPACITY_MAX
} mpe_CcOpacity;
```

where

`MPE_CC_OPACITY_EMBEDDED`

indicates the opacity embedded in the stream.

`MPE_CC_OPACITY_TRANSPARENT`

indicates transparent opacity. Transparent means it is invisible to the viewer. The underlying video is not blended or filtered.

`MPE_CC_OPACITY_TRANSLUCENT`

indicates translucent opacity. Translucent means it is blended (or filtered) with the underlying video. The amount of translucence is defined by the hardware. It is implementation-specific.

`MPE_CC_OPACITY_SOLID`

indicates solid opacity. Underlying video is not visible.

`MPE_CC_OPACITY_FLASHING`

indicates flashing opacity. Flashing frequency is implementation-specific.

`MPE_CC_OPACITY_MAX`

specifies the maximum value that can be stored in `mpe_CcOpacity`.

mpe_CcState

`mpe_CcState` specifies the state of the closed captioning. `mpe_CcState` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcState {
    MPE_CC_STATE_OFF,
    MPE_CC_STATE_ON,
    MPE_CC_STATE_ON_MUTE,
    MPE_CC_STATE_MAX
} mpe_CcState;
```

where

`MPE_CC_STATE_OFF`

indicates the closed captioning is off.

`MPE_CC_STATE_ON`

indicates the closed captioning is on.

`MPE_CC_STATE_ON_MUTE`

indicates closed captioning displays only when the user audio is muted and hidden when the audio is unmuted.

`MPE_CC_STATE_MAX`

specifies the maximum value that can be stored in `mpe_CcState`.

mpe_CcTextStyle

`mpe_CcTextStyle` determines if the text has style attributes for `mpe_CcAttributes fontItalic` and `mpe_CcAttributes fontUnderline`. For example, if the `fontUnderline` is set to `MPE_CC_TEXT_STYLE_TRUE`, the closed-captioning text displays underlined. If the `fontUnderline` is set to `MPE_CC_TEXT_STYLE_FALSE`, the closed-captioning text is not underlined. `mpe_CcTextStyle` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcTextStyle {
    MPE_CC_TEXT_STYLE_EMBEDDED_TEXT = -1,
    MPE_CC_TEXT_STYLE_FALSE,
    MPE_CC_TEXT_STYLE_TRUE,
    MPE_CC_TEXT_STYLE_MAX
} mpe_CcTextStyle;
```

where

`MPE_CC_TEXT_STYLE_EMBEDDED_TEXT`

indicates an embedded text style.

`MPE_CC_TEXT_STYLE_FALSE`

indicates the text does not have any style attribute.

`MPE_CC_TEXT_STYLE_TRUE`

indicates the text has style attributes.

`MPE_CC_TEXT_STYLE_MAX`

specifies the maximum value that can be stored in `mpe_CcTextStyle`.

mpe_CcType

`mpe_CcType` specifies the type of signal. `mpe_CcType` is defined in `mpeos_caption.h` as follows:

```
typedef enum mpe_CcType {  
    MPE_CC_TYPE_ANALOG,  
    MPE_CC_TYPE_DIGITAL,  
    MPE_CC_TYPE_MAX  
} mpe_CcType;
```

where

`MPE_CC_TYPE_ANALOG`
indicates analog signal.

`MPE_CC_TYPE_DIGITAL`
indicates digital signal.

`MPE_CC_TYPE_MAX`
specifies the maximum value that can be stored in `mpe_CcType`.

mpe_Error

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

Supported functions

The following closed-captioning functions need to be supported:

- `mpeos_ccGetAnalogServices`
gets the current analog service map.
- `mpeos_ccGetAttributes`
gets the current closed-captioning attributes.
- `mpeos_ccGetCapability`
gets a list of supported values for a given attribute.
- `mpeos_ccGetClosedCaptioning`
gets the current state of closed captioning.
- `mpeos_ccGetDigitalServices`
gets the current digital closed-captioning service map.
- `mpeos_ccGetSupportedServiceNumbers`
gets the services which support closed captioning.
- `mpeos_ccSetAnalogServices`
sets the analog closed-captioning services.
- `mpeos_ccSetAttributes`
sets the closed-captioning attributes.
- `mpeos_ccSetClosedCaptioning`
sets the state of the closed captioning.
- `mpeos_ccSetDigitalServices`
sets the digital closed-captioning service.

mpeos_ccGetAnalogServices

gets the current analog service map.

syntax mpe_Error mpeos_ccGetAnalogServices(
 mpe_CcAnalogServiceMap * service);

parameter(s) *service* is an output pointer for returning the current analog service map. *mpe_CcAnalogServiceMap* is defined in the *mpe_CcAnalogServiceMap* section.

value returned If the call is successful, *mpeos_ccGetAnalogServices()* should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_CC_ERROR_INVALID_PARAM
 indicates an invalid parameter.

description *mpeos_ccGetAnalogServices()* should return the current analog closed-captioning service map.

related function(s) *mpeos_ccSetAnalogServices*

mpeos_ccGetAttributes

gets the current closed-captioning attributes.

syntax mpe_Error mpeos_ccSetAttributes(
 mpe_CcAttributes * *attribute*,
 mpe_CcType *ccType*);

parameter(s) *attribute* is an output pointer to an *mpe_CcAttributes* data structure.
mpe_CcAttributes is described in the *mpe_CcAttributes* section.

ccType specifies the type of signal for the selected attribute.
mpe_CcType is described in the *mpe_CcType* section.
Currently, the following values are supported for *ccType*:

MPE_CC_TYPE_ANALOG
 specifies an analog signal.

MPE_CC_TYPE_DIGITAL
 specifies a digital signal.

value returned If the call is successful, *mpeos_ccGetAttributes()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_CC_ERROR_INVALID_PARAM
 indicates an invalid input parameter.

description *mpeos_ccGetAttributes()* should get the current attributes for the closed-captioning text, for either digital or analog closed captioning, depending on the value of the *ccType* parameter.

related function(s) *mpeos_ccSetAttributes*

mpeos_ccGetCapability

gets a list of supported values for a given attribute.

syntax mpe_Error mpeos_ccGetCapability(
 mpe_CcAttribType attrib,
 mpe_CcType type,
 void ** value,
 uint32_t * size);

parameter(s) attrib

specifies a single attribute for which the capabilities are returned. *attrib* is returned by a previous call to `mpeos_ccSetAttributes()`. `mpe_CcAttribType` is described in the *mpe_CcAttribType* section. Currently, the following values are supported for *attrib*:

MPE_CC_ATTRIB_FONT_COLOR
 specifies the color for the caption characters.

MPE_CC_ATTRIB_BACKGROUND_COLOR
 specifies the background color for the caption.

MPE_CC_ATTRIB_FONT_OPACITY
 specifies opacity for the caption characters.

MPE_CC_ATTRIB_BACKGROUND_OPACITY
 specifies opacity for the caption background.

MPE_CC_ATTRIB_FONT_STYLE
 specifies the font style.

MPE_CC_ATTRIB_FONT_SIZE
 specifies the point size of the font.

MPE_CC_ATTRIB_FONT_ITALIC
 specifies the italic font.

MPE_CC_ATTRIB_FONT_UNDERLINE
 specifies the underlined font.

MPE_CC_ATTRIB_BORDER_TYPE
 specifies a border type.

MPE_CC_ATTRIB_BORDER_COLOR
 specifies a color border.

MPE_CC_ATTRIB_WIN_COLOR
 specifies a color window.

MPE_CC_ATTRIB_WIN_OPACITY
 specifies an opaque window.

type specifies the type of signal in which the attributes will be used. *mpe_CcType* is described in the *mpe_CcType* section. Currently, the following values are supported for *type*:

MPE_CC_TYPE_ANALOG
specifies an analog signal.

MPE_CC_TYPE_DIGITAL
specifies a digital signal.

value is an output pointer to return the list of possible values for the *attrib* parameter. If the underlying platform does not support the attribute, the pointer is set to NULL.

size is an output pointer to return the number of entries in the list returned by the *value* parameter. If the underlying platform does not support the attribute, the pointer is set to 0.

value returned If the call is successful, *mpeos_ccGetCapability()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_CC_OSERR indicates an error in the operating system.

description *mpeos_ccGetCapability()* should get a list of all supported capabilities for an attribute. The capability is retrieved for digital or analog closed captioning, depending on the value of the *type* parameter.

related function(s) *mpeos_ccGetAttributes*
mpeos_ccSetAttributes

mpeos_ccGetClosedCaptioning

gets the current state of closed captioning.

syntax mpe_Error mpeos_ccGetClosedCaptioning(mpe_CcState * state);

parameter(s) *state* is an output pointer to the current closed-captioning state. *mpe_CcState* is described in the *mpe_CcState* section. Currently, the following values are supported for *state*:

MPE_CC_STATE_OFF
indicates the closed captioning is off.

MPE_CC_STATE_ON
indicates the closed captioning is on.

MPE_CC_STATE_ON_MUTE
indicates closed captioning displays only when the user audio is muted and hidden when the audio is unmuted.

value returned If the call is successful, `mpeos_ccGetClosedCaptioning()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

MPE_CC_ERROR_INVALID_PARAM
indicates an invalid input parameter.

description `mpeos_ccGetClosedCaptioning()` should return the current closed-captioning state.

related function(s) `mpeos_ccSetClosedCaptioning`

mpeos_ccGetDigitalServices

gets the current digital closed-captioning service map.

syntax mpe_Error mpeos_ccGetDigitalServices(
 mpe_CcDigitalServiceMap * service);

parameter(s) *service* is an output pointer for returning the current digital closed-captioning map. *mpe_CcDigitalServiceMap* is defined in the *mpe_CcDigitalServiceMap* section.

value returned If the call is successful, *mpeos_ccGetDigitalServices()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:
MPE_CC_ERROR_INVALID_PARAM
 indicates an invalid parameter.

description *mpeos_ccGetDigitalServices()* should return the current digital closed-captioning service map.

related function(s) *mpeos_ccSetDigitalServices*

mpeos_ccGetSupportedServiceNumbers

gets the services which support closed captioning.

syntax mpe_Error mpeos_ccGetSupportedServiceNumbers(
 uint32_t ** servicesArray,
 uint32_t * servicesArraySize);

parameter(s) *servicesArray* is an output pointer to the address of a pointer to an array containing the supported services.

servicesArraySize is an output pointer to the address of an integer buffer containing the array size of the supported services.

value returned If the call is successful, mpeos_ccGetSupportedServiceNumbers() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_CC_ERROR_INVALID_PARAM
indicates an invalid input parameter.

description mpeos_ccGetSupportedServiceNumbers() should get a list of the analog and digital service numbers that support closed captioning.

related function(s) none

mpeos_ccSetAnalogServices

sets the analog closed-captioning services.

syntax `mpe_Error mpeos_ccSetAnalogServices(
 uint32_t service);`

parameter(s) `service` specifies the desired analog closed-caption service. For more information on analog services, refer to the `mpe_CcAnalogServices` section. The following OCAP-defined values are supported for `service`:

`MPE_CC_ANALOG_SERVICE_NONE`
specifies no analog service.

`MPE_CC_ANALOG_SERVICE_CC1`
specifies the primary synchronous caption service.

`MPE_CC_ANALOG_SERVICE_CC2`
specifies the special non-synchronous channel.

`MPE_CC_ANALOG_SERVICE_CC3`
specifies the secondary synchronous caption service.

`MPE_CC_ANALOG_SERVICE_CC4`
specifies the secondary special non-synchronous channel.

`MPE_CC_ANALOG_SERVICE_T1`
specifies the first text service.

`MPE_CC_ANALOG_SERVICE_T2`
specifies the second text service.

`MPE_CC_ANALOG_SERVICE_T3`
specifies the third text service.

`MPE_CC_ANALOG_SERVICE_T4`
specifies the fourth text service.

value returned If the call is successful, `mpeos_ccSetAnalogServices()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_CC_ERROR_INVALID_PARAM`
indicates an invalid input parameter or the parameter is set to NULL.

`MPE_CC_ERROR_OS`
indicates an operating system error occurred.

description `mpeos_ccSetAnalogServices()` should set the analog closed-captioning service to the one specified by `service`. `service` can take one of the value defined by `mpe_CcAnalogServices`.

related function(s) `mpeos_ccGetAnalogServices`

mpeos_ccSetAttributes

sets the closed-captioning attributes.

syntax mpe_Error mpeos_ccSetAttributes(
 mpe_CcAttributes * *attribute*,
 uint16_t *type*,
 mpe_CcType *ccType*);

| | | |
|---------------------|------------------|---|
| parameter(s) | <i>attribute</i> | is an input pointer to the <i>mpe_CcAttributes</i> data structure containing the user preferences for displaying the closed-captioning text. To use the embedded values in the closed-captioning stream, the attribute must be set to the embedded value (for example, to set embedded color, use the <i>MPE_EMBEDDED_COLOR</i>). Only the values returned by <i>mpeos_ccGetCapabilities()</i> can be set. If <i>attribute</i> is NULL, all attributes are reset to embedded values. <i>mpe_CcAttributes</i> is described in the <i>mpe_CcAttributes</i> section. |
| | <i>type</i> | specifies the attribute or attributes to change to the setting in <i>attribute</i> data structure. Currently, the following bit-field settings are supported: |
| | | MPE_CC_ATTRIB_FONT_COLOR specifies the color for the caption characters. |
| | | MPE_CC_ATTRIB_BACKGROUND_COLOR specifies the background color for the caption. |
| | | MPE_CC_ATTRIB_FONT_OPACITY specifies the opacity for the caption characters. |
| | | MPE_CC_ATTRIB_BACKGROUND_OPACITY specifies the background opacity for the caption. |
| | | MPE_CC_ATTRIB_FONT_STYLE specifies the font style. |
| | | MPE_CC_ATTRIB_FONT_SIZE specifies the point size of the font. |
| | | MPE_CC_ATTRIB_FONT_ITALIC specifies an italic font. |
| | | MPE_CC_ATTRIB_FONT_UNDERLINE specifies an underlined font. |
| | | MPE_CC_ATTRIB_BORDER_TYPE specifies the border type. |
| | | MPE_CC_ATTRIB_BORDER_COLOR specifies the border color. |
| | | MPE_CC_ATTRIB_WIN_COLOR specifies the window color. |

MPE_CC_ATTRIB_WIN_OPACITY
specifies the window opacity.

ccType specifies the type of signal for which the attributes will be used. *mpe_CcType* is described in the *mpe_CcType* section. Currently, the following values are supported for *ccType*:

MPE_CC_TYPE_ANALOG
specifies an analog signal.

MPE_CC_TYPE_DIGITAL
specifies a digital signal.

value returned If the call is successful, *mpeos_ccSetAttributes()* should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_CC_ERROR_INVALID_PARAM
indicates an invalid input parameter or the parameter is set to NULL.

MPE_CC_ERROR_OS
indicates an operating system error occurred.

description *mpeos_ccSetAttributes()* should set the user-specified attributes for displaying the closed-captioning text. These attributes are set for digital or analog closed captioning, depending on the value of the *ccType* parameter.

related function(s) *mpeos_ccGetAttributes*

mpeos_ccSetClosedCaptioning

sets the state of the closed captioning.

syntax mpe_Error mpeos_ccSetClosedCaptioning(mpe_CcState state);

parameter(s) *state* indicates the new state of closed captioning. *mpe_CcState* is described in the *mpe_CcState* section. Currently, the following values are supported for *state*:

MPE_CC_TURN_OFF

indicates closed captioning is off.

MPE_CC_TURN_ON

indicates closed captioning is on.

MPE_CC_TURN_ON_MUTE

indicates closed captioning displays when the user's audio is muted and is hidden when the audio is unmuted.

value returned If the call is successful, `mpeos_ccSetClosedCaptioning()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

MPE_CC_ERROR_OS

indicates an operating system error occurred.

description `mpeos_ccSetClosedCaptioning()` should set the state of the closed captioning.

related function(s) `mpeos_ccGetClosedCaptioning`

mpeos_ccSetDigitalServices

sets the digital closed-captioning service.

syntax mpe_Error mpeos_ccSetDigitalServices(
 uint32_t *service*);

parameter(s) *service* specifies a digital closed-caption service as defined by OCAP. To set a digital *service* select a number between 1 and 63. Otherwise, select MPE_CC_DIGITAL_SERVICE_NONE.

value returned If the call is successful, mpeos_ccSetDigitalServices() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_CC_ERROR_OS
 indicates an operating system error occurred.

description mpeos_ccSetDigitalServices() should set digital closed-captioning service.

related function(s) mpeos_ccGetDigitalServices

CDL Common-Download API

Overview

The common-download Application Programming Interface (API) is used by the OpenCable Application Platform (OCAP) stack to manage the OpenCable download process. Common download is used to download new code to the set-top box through the CableCARD. For more information, refer to the CableCard Interface 2.0 Specification.

Before reading this chapter, you should be familiar with:

- ◆ the OpenCable common-download specification
- ◆ the OCAP specification

After reading this chapter, you should be able to port the OCAP required functions for common download

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the common-download functions:

`MPE_EINVAL`

`MPE_SUCCESS`

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPE_EINVAL`

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

`MPE_SUCCESS`

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types, which are defined in `mpe_error.h` and `mpeos_event.h`, are used by the common-download functions:

`mpe_Error`

`mpe_EventQueue`

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

Events

The following decoding event, which is defined in `mpe_cd1.h`, is used by the common-download functions:

```
MPE_CD1_EVENT_SHUTDOWN
```

MPE_CD1_EVENT_SHUTDOWN

`MPE_CD1_EVENT_SHUTDOWN` specifies the listener is no longer used. When a new listener registers for `mpeos_cd1Register()`, or when a queue is unregistered, an `MPE_CD1_EVENT_SHUTDOWN` is sent to the previously registered queue. `MPE_CD1_EVENT_SHUTDOWN` is defined as follows:

```
#define MPE_CD1_EVENT_SHUTDOWN ((mpe_Event)(-1))
```

- ◆ **NOTE:** `MPE_CD1_EVENT_SHUTDOWN` is sent from MPEOS to the OCAP layer and used as a typeCode for `org.ocap.system.event.DeferredDownloadEvent` download event .
-

Supported functions

The following common-download functions need to be supported:

`mpeos_cd1Register`
register to receive common-download events.

`mpeos_cd1StartDownload`
initiates the common-download process.

`mpeos_cd1Unregister`
unregisters from receiving common-download events.

mpeos_cdlRegister

register to receive common-download events.

syntax `mpe_Error mpeos_cdlRegister(
 mpe_EventQueue queueId,
 void * act);`

parameter(s) `queueId` identifies the queue in which to send the events associated with this download. `mpe_EventQueue` is described in the `mpe_EventQueue` section. Currently, the following event is supported:

`MPE_CDL_EVENT_SHUTDOWN`

specifies the listener is no longer used.

`act` is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous download events are sent to the queue specified in `queueId` (via `mpeos_eventQueueSend()`), the event should pass this `act` pointer in the *optionalEventData2* field.

value returned If the call is successful, `mpeos_cdlRegister()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates `queueId` is not a valid queue.

description `mpeos_cdlRegister()` should unregister the previous event queue by sending a `MPE_CDL_EVENT_SHUTDOWN` event to the queue through `mpeos_eventQueueSend()` and then register `queueId` as the listener for asynchronous common-download events.

- ◆ **NOTE:** Only one asynchronous event listener is supported. Subsequent call to `mpeos_cdlRegister()` replaces the previous listener.

related function(s) `mpeos_cdlUnregister`

mpeos_cd1StartDownload

initiates the common-download process.

syntax mpe_Error mpeos_cd1StartDownload(void);

parameter(s) none

value returned If the call is successful, mpeos_cd1StartDownload() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates there is no registered listener.

description mpeos_cd1StartDownload() should initiate the common-download process and notify the current listener queue that a download event has started.

- ◆ **NOTE:** In most cases mpeos_cd1StartDownload() will not return, because the common-download call is normally used to update the software and reset the set-top box.

related function(s) mpeos_cd1Register

mpeos_cdlUnregister

unregisters from receiving common-download events.

syntax `mpe_Error mpeos_cdlUnregister(
 mpe_EventQueue queueId,
 void * act);`

parameter(s) `queueId` identifies the queue to be unregistered. `mpe_EventQueue` is described in the *mpe_EventQueue* section. Currently, the following event is supported:

`MPE_CDL_EVENT_SHUTDOWN`
specifies the listener is no longer used.

`act` is an input pointer to the ACT used to register the listener. Both `queueId` and `act` must match the parameters used when the listener was registered.

value returned If the call is successful, `mpeos_cdlUnregister()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates one or both of the parameters do not match the current listener.

description `mpeos_cdlUnregister()` should unregister the current event queue from receiving notification of asynchronous common-download events by sending a `MPE_CDL_EVENT_SHUTDOWN` event to the queue through `mpeos_eventQueueSend()`.

related function(s) `mpeos_cdlRegister`

DBG Debug API

Overview

The debug Application Programming Interface (API) is used by the Multimedia Platform Environment (MPE) and parts of the OpenCable Application Platform (OCAP) stack to provide basic status and diagnostic information messages to the user/developer. These messages are generated in debug builds of the MPE stack only, and are removed for release builds.

The exact mechanism of the debug output is platform dependent. On a set-top box, this could be in the form of console output messages, socket communication to a logging server, or additions to a logging file.

The debug APIs offer the ability to enable and disable logging for individual modules and logging levels at boot-time and run-time. It provides both internal and external developers using the OCAP stack more control of logging output in debugging builds.

Before reading this chapter, you should be familiar with:

- ◆ how debugging works
- ◆ the debug OCAP requirements

After reading this chapter, you should be able to port the debug functionality within the OCAP stack

Logging

Logging is done at the Java level and the native C level. The Java level uses the `log4j.properties` file to define and initialize logging. The native C level uses the `mpeenv.ini` file to define and initialize logging. Because the two levels need to be logged separately, consider the Java Virtual Machine (JVM) as the dividing element. The native C level provides services and interfaces to the Java portion of the OCAP stack.

Java-level logging

Java-level logging funnels all the OCAP stack logging through the `Log4j` logging utility, which maps into a custom native logging module (the “JAVA” module). All `Log4j` logging methods are wrapped by a constant conditional, which causes the entire logging method call and string to be removed from a production build (enhancing performance and saving code-size).

Native C level logging

Native C level logging uses a set of macros, `MPE_LOG()` and `MPEOS_LOG()`. These macros wrap calls to the native MPE and Multimedia Platform Environment Operating System (MPEOS) logging functions. The macros are used so that additional functionality can be added in future releases (for example, selectively removing unwanted logging calls and strings in debugging builds, faster run-time determination of the need to call the logging print function, etc.).

In addition to wrapping calls to `mpe_dbgMsg()` and `mpeos_dbgMsg()` within the `MPE_LOG()` and `MPEOS_LOG()` macros, a module parameter and a logging level parameter have been added to all native C logging messages. So, the new syntax looks like:

```
MPE_LOG(int level, int modname, const char *msg, ...)  
MPEOS_LOG(int level, int modname, const char *msg, ...)
```

As with the rest of the native OCAP stack, the MPE version of the macro must be used by all external users of the MPE APIs (for example, by Esmertec Virtual Machine (EVM), Java Native Interface (JNI), DirectFB, etc.). The macro always goes through the `mpe_dbgMsg()` function table pointer. However, internal MPE/MPEOS code can directly use the MPEOS version for speed as it can directly map into a function call (without going through the MPE function table).

Logging commands

The MPE log command allows you to query the logging levels enabled in a particular module. The syntax is as follows:

promptmpe log command module logging level

where

prompt refers to the prompt displayed by the set-top box.

command specifies the command to execute. Currently, the following values are defined for *command*:

get gets the current values.

set sets the current values.

module specifies the module.

logging level specifies the type of logging.

Logging examples

The module and logging level references (`LOG.MPE.module=logging level`) must be formatted correctly for the logging to work properly. Modules must be configured in the `mpeenv.ini` using the following rules:

- ◆ use only uppercase characters
- ◆ use at least one space between each module
- ◆ use the correct spelling
- ◆ do not use and extraneous characters (\$, %, etc.)

If any errors occur in the formatting when logging is parsing the configuration, the request is ignored.

At boot-time the configuration of logging is evaluated in the `mpeenv.ini` file. Each module may have an entry in the `mpeenv.ini` file. Each entry is configured using a set of space delimited keywords referring to the various logging levels. The presence of a keyword enables logging for that level and only that level. To disable logging, preface the keywords with an exclamation mark (!).

Example 1, track only the threading module:

```
LOG.MPE.DEFAULT=NONE
LOG.MPE.THREAD=DEBUG
```

Example 2, turn off debug logging in memory module:

```
LOG.MPE.DEFAULT=ALL
LOG.MPE.MEM=!DEBUG
```

Example 3, turn on everything except INFO and DEBUG:

```
LOG.MPE.DEFAULT=ALL !INFO !DEBUG
```

Example 4, enable only trace 1 and 8 using the error and fatal logging levels:

```
LOG.MPE.SYS=FATAL ERROR !TRACE TRACE1 TRACE8
```

Definitions

The following definition, which is defined in `mpeos_debug.h`, is used by the debug functions:

`MPEOS_LOG`

-
- ◆ **NOTE:** The actual definition is included here for informational purposes only. You should not need to change the value of the definition, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPEOS_LOG`

`MPEOS_LOG` controls the inclusion of logging at compile time. `MPEOS_LOG` is defined as follows:

```
#define MPEOS_LOG OS_MPEOS_LOG
```

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the debug functions:

`MPE_EINVAL`

`MPE_SUCCESS`

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPE_EINVAL`

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

`MPE_SUCCESS`

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_dbg.h`, `mpe_types.h`, and `mpeos_event.h`, are used by the debug functions:

| | |
|-----------------------------|----------------------------------|
| <code>mpe_Bool</code> | <code>mpe_DbgStatusFormat</code> |
| <code>mpe_DbStatusId</code> | <code>mpe_DbStatusType</code> |
| <code>mpe_Error</code> | <code>mpe_EventQueue</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_DbgStatusFormat`

`mpe_DbgStatusFormat` specifies the format of status notifications. The MPE status formats are defined in `mpe_dbg.h`. `mpe_DbgStatusFormat` is defined in `mpe_dbg.h` as follows:

```
typedef enum {
    MPE_DBG_STATUS_FMT_ANY = 0,
    MPE_DBG_STATUS_FMT_INT = 1,
    MPE_DBG_STATUS_FMT_STRING = 2,
    MPE_DBG_STATUS_FMT_BEAN = 3
} mpe_DbgStatusFormat;
```

`mpe_DbStatusId`

`mpe_DbStatusId` specifies the status identifier. `mpe_DbStatusId` is composed of [`mpe_LogModule` << 16 | `mpe_DbStatusTypeId`]. `mpe_DbStatusId` is defined in `mpe_dbg.h` as follows:

```
typedef unsigned int mpe_DbStatusId;
```

`mpe_DbStatusType`

`mpe_DbStatusType` defines the association between a status type string and its integer identifier. `mpe_DbStatusType` is defined in `mpe_dbg.h` as follows:

```
typedef struct _mpe_DbStatusType mpe_DbStatusType;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

mpe_logModule

`mpe_logModule` is used to identify the source module of an MPE_LOG operation. The source module is used at runtime to assist with filtering of log messages. The values defined by `mpe_logModule` are also used in construction of two additional associated tables. The first table, `mpeos_g_LogControlTb1`, is used to hold the logging control information associated with each module. The second table, which is located in `%OCAPROOT%\mpe\os\common\mpeos_dbg_log.c` is used for mapping the module identifiers to their associated module string values. Any updates to `mpe_logModule` needs to be reflected in both of these additional tables. Currently, `mpe_logModule` is defining `mpe_dbg.h` as follows:

```
typedef enum {
    ENUM_MPE_MOD_BEGIN = 0,
    MPE_MOD_TARGET = ENUM_MPE_MOD_BEGIN,
    MPE_MOD_CC,
    MPE_MOD_CDL,
    MPE_MOD_COND,
    MPE_MOD_DBG,
    MPE_MOD_DIRECTFB,
    MPE_MOD_DISP,
    MPE_MOD_DLL,
    MPE_MOD_DVR,
    MPE_MOD_EVENT,
    MPE_MOD_ED,
    MPE_MOD_FILESYS,
    MPE_MOD_FILTER,
    MPE_MOD_FP,
    MPE_MOD_GFX,
    MPE_MOD_JAVA,
    MPE_MOD_JNI,
    MPE_MOD_JNI_AWT,
    MPE_MOD_JVM,
    MPE_MOD_MEDIA,
    MPE_MOD_MEM,
    MPE_MOD_MUTEX,
    MPE_MOD_NET,
    MPE_MOD_OS,
    MPE_MOD_POD,
    MPE_MOD_SI,
    MPE_MOD_SOUND,
    MPE_MOD_SYS,
    MPE_MOD_TEST,
    MPE_MOD_THREAD,
    MPE_MOD_UTIL,
    MPE_MOD_UI,
    MPE_MOD_MRDRV,
    MPE_MOD_SNMP,
```

```

        MPE_MOD_STORAGE,
        MPE_MOD_PROD,
        MPE_MOD_FDR,
        ENUM_MPE_MOD_COUNT
    } mpe_logModule;

```

where

| | |
|---------------------------|---|
| ENUM_MPE_MOD_BEGIN | specifies the beginning of the array index. The array index is initialized as 0. |
| MPE_MOD_TARGET | specifies the port-specific to the target device. The target device is initialized as ENUM_MPE_MOD_BEGIN . |
| MPE_MOD_CC | specifies closed captioning API logging. |
| MPE_MOD_CDL | specifies common download API logging. |
| MPE_MOD_COND | specifies logging of conditionals. |
| MPE_MOD_DBG | specifies debug API logging. |
| MPE_MOD_DIRECTFB | specifies DirectFB logging. |
| MPE_MOD_DISP | specifies display API logging. |
| MPE_MOD_DLL | specifies Dynamic Link Library (DLL) management API logging. |
| MPE_MOD_DVR | specifies Digital Video Recorder (DVR) API logging. |
| MPE_MOD_EVENT | specifies event API logging. |
| MPE_MOD_ED | specifies event dispatch API logging. |
| MPE_MOD_FILESYS | specifies file system API logging. |
| MPE_MOD_FILTER | specifies section filtering API logging. |
| MPE_MOD_FP | specifies front panel API logging. |
| MPE_MOD_GFX | specifies graphics API logging. |
| MPE_MOD_JAVA | specifies Java code logging. |
| MPE_MOD_JNI | specifies JNI logging |
| MPE_MOD_JNI_AWT | specifies AWT implementation (JNI) logging. |
| MPE_MOD_JVM | specifies JVM logging. |
| MPE_MOD_MEDIA | specifies media API logging. |
| MPE_MOD_MEM | specifies memory API logging. |
| MPE_MOD_MUTEX | specifies thread synchronization API logging. |

MPE_MOD_NET specifies network API logging.
MPE_MOD_OS specifies operating system API logging.
MPE_MOD_POD specifies Point-of-Deployment (POD) API logging.
MPE_MOD_SI specifies service information API logging.
MPE_MOD_SOUND specifies sound playback logging
MPE_MOD_SYS specifies system manager logging.
MPE_MOD_TEST specifies MPE testing framework logging.
MPE_MOD_THREAD
 specifies thread API logging.
MPE_MOD_UTIL specifies utility API logging.
MPE_MOD_UI specifies user interface logging.
MPE_MOD_MRDRV specifies multiple-room DVR logging.
MPE_MOD_SNMP specifies SNMP logging.
MPE_MOD_STORAGE
 specifies storage logging.
MPE_MOD_PROD specifies production logging module logging.
MPE_MOD_FDR specifies FDR logging.
ENUM_MPE_MOD_COUNT
 specifies the number of modules specified for logging.

In addition there is a single alias, DEFAULT, which is evaluated first (regardless of the order it appears in the `mpeenv.ini` file). The DEFAULT module pre-configures all modules to match its settings. Then, each additional module is able to modify those settings.

mpeos_g_logControlTbl

`mpeos_g_logControlTbl[]` is a compact encoding of the logging configuration specified in the `mpeenv.ini` file. This encoding is intended to be opaque. Consequently, the macro `WANT_LOG` should be used in implementations of `mpeos_dbgMsgRaw()` to determine whether the desired logging type is enabled for the specified module. `mpeos_dbgMsgRaw()` outputs the specified debug message to the target debug output (for example, console, database). The macro `WANT_LOG` returns zero if logging is not enabled for the specified type and module, it returns non-zero otherwise.

- ◆ **CAUTION:** `mpeos_g_logControlTbl[]` is a global variable. Modifications of this variable at runtime will affect all consequent logging. There is no concurrent access protection for the table, thus care must be taken when allowing runtime modification of the table in such an environment (for example, threaded, multi-process).
- ◆ **NOTE:** This implementation was designed to maximize efficiency in configuration lookups.

`mpeos_g_logControlTbl[]` is defined in `mpeos_dbg.h` as follows:

```
extern uint32_t mpeos_g_logControlTbl[ module ];
```

Supported functions

The following debug functions need to be supported:

`dbg_logViaUDP`

determines whether logging uses the UDP.

`mpeos_dbgLogUDP`

sends a log message over UDP.

`mpeos_dbgStatusGetTypes`

acquires the status types supported by the port-level APIs.

`mpeos_dbgStatusRegister`

registers an event queue for status events.

`mpeos_dbgStatusRegisterInterest`

registers for delivery of a specific status event.

`mpeos_dbgStatusUnregister`

unregisters for status events.

`mpeos_dbgStatusUnregisterInterest`

unregisters for notification of specific status event.

dbg_logViaUDP

determines whether logging uses the UDP.

syntax `extern mpe_Bool dbg_logViaUDP;`

parameter(s) none

value returned none

description `dbg_logViaUDP()` should determine whether or not the logging uses the User Datagram Protocol (UDP). If TRUE, the logging uses UDP. If FALSE, UDP is not used.

related function(s) `mpeos_dbgLogUDP`

mpeos_dbgLogUDP

sends a log message over UDP.

syntax void mpeos_dbgLogUDP(
 const char * *format*,
 va_list *args*);

parameter(s) *format* is an input pointer to a string containing a print format (for example, printf).

args specifies the arguments for the type of format.

value returned none

description mpeos_dbgLogUDP() should send a log message over UDP.

related function(s) none

mpeos_dbgStatusGetTypes

acquires the status types supported by the port-level APIs.

syntax mpe_Error mpeos_dbgStatusGetTypes(mpe_DbgStatusType * types);

parameter(s) *types* is an input pointer to the global debug status type table to further populate with the port-level status types (see table `mpeDbgStatusTypes` in `mpe_dbg.h` for the location to populate and the space available for port-level types). `mpe_DbgStatusType` is described in the *mpe_DbgStatusType* section.

value returned If the call is successful, `mpeos_dbgStatusGetTypes()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates *types* has an invalid value.

description `mpeos_dbgStatusGetTypes()` should acquire the status types supported by the port-level APIs and stores them in a table specified by the *types* pointer.

This function should populate the `mpeDbgStatusTypes` table with the values supported in the porting layer. Add port-level status types to the `mpeDbgStatusTypes` table located in `mpe_dbg.h`.

related function(s) none

mpeos_dbgStatusRegister

registers an event queue for status events.

syntax mpe_Error mpeos_dbgStatusRegister(
 mpe_EventQueue *queueId*,
 void * *act*);

parameter(s) *queueId* identifies the queue in which to send the events associated with status changes. *mpe_EventQueue* is described in the *mpe_EventQueue* section.

act is an input pointer to an Asynchronous Completion Token (ACT) to deliver with the event. When events are sent to the queue specified in *queueId* (via *mpeos_eventQueueSend()*), the event send should pass this *act* pointer in the *optionalEventData2* field.

value returned If the call is successful, *mpeos_dbgStatusRegister()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_dbgStatusRegister()* should register one event queue, and only one queue, for status events. Currently, *mpeos_dbgStatusRegister()* is the framework for future implementations that need a delivery system to receive debug events (for example, VOD status events).

◆ **NOTE:** Currently, no events are defined in the OCAP stack. Future ports should define the type of events needed.

related function(s) *mpeos_dbgStatusUnregister*

mpeos_dbgStatusRegisterInterest

registers for delivery of a specific status event.

syntax mpe_Error mpeos_dbgStatusRegisterInterest(
 mpe_DbгStatusId *typeid*,
 mpe_DbгStatusFormat *format*,
 void * *param*);

parameter(s) *typeid* specifies the port-specific type of status to monitor (for example, string, object, integer). *mpe_DbгStatusId* is described in the *mpe_DbгStatusId* section.

format specifies the status format identifier. *mpe_DbгStatusFormat* is described in the *mpe_DbгStatusFormat* section.

param is an input pointer for an optional parameter for the particular status event.

value returned If the call is successful, *mpeos_dbgStatusRegisterInterest()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:
MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_dbgStatusRegisterInterest()* should register for delivery of a specific status event.

◆ **NOTE:** Currently, no types are defined in the OCAP stack. Future ports should define the types needed.

related function(s) *mpeos_dbgStatusUnregisterInterest*

mpeos_dbgStatusUnregister

unregisters for status events.

syntax mpe_Error mpeos_dbgStatusUnregister(mpe_EventQueue *queueId*);

parameter(s) *queueId* identifies the queue to unregister. mpe_EventQueue is described in the *mpe_EventQueue* section.

value returned If the call is successful, mpeos_dbgStatusUnregister() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *queueId* has an invalid value.

description mpeos_dbgStatusUnregister should unregister the event queue.

related function(s) mpeos_dbgStatusRegister

mpeos_dbgStatusUnregisterInterest

unregisters for notification of specific status event.

syntax mpe_Error mpeos_dbgStatusUnregisterInterest(
 mpe_DbгStatusId *typeld*);

parameter(s) *typeld* specifies the port-specific type of status to monitor (for example, string, object, integer). *mpe_DbгStatusId* is described in the *mpe_DbгStatusId* section.

value returned If the call is successful, *mpeos_dbgStatusUnregisterInterest()* should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *typeld* has an invalid value.

description *mpeos_dbgStatusUnregisterInterest* should unregisters for notification of a port-specific status event.

related function(s) *mpeos_dbgStatusRegisterInterest*

DSP Display API

Overview

The display Application Programming Interface (API) is used by the Multimedia Platform Environment (MPE) and parts of the OpenCable Application Platform (OCAP) stack to provide basic discovery and configuration of the display including (potential) support for multiple screens, multiple devices (background, graphics, and video), and multiple device configurations. Also provided are functions to decode Moving Picture Experts Group (MPEG)-encoded still images (known as I-Frames) from memory (`mpeos_dispBGImageNew()` and `mpeos_dispDisplayBGImage()`). Currently, only a single screen is supported.

Before reading this chapter, you should be familiar with:

- ♦ display requirements

After reading this chapter, you should be able to port the display functionality within the OCAP stack

Background and video reference model

The Display API provides functions for the background and video planes. The graphics plane functionality is provided by the Graphics API. This section provides information about the following topics:

Screen layout

Coordinate system

Screen layout

The screen order from back to front is background, video, graphics. However, there may be more than one device of each type (for example, 1 background, 2 video, and 1 graphics).

Background

The background plane typically displays one background color, but on some platforms an image can be displayed. The background plane is always full-screen and is always combined with the video plane using a source rule. If the video plane is full-screen, the background plane is not visible.

The minimum OCAP requirement for standard-definition background resolution is 720x480 or 640x480. The minimum OCAP requirement for high-definition background resolution, in addition to supporting 720x480 or 640x480 for standard, is 1920x1080.

Graphics

The graphics plane is used for on-screen graphics, for example, subtitles, on-screen menus, Emergency Alert System (EAS), and application graphics.

The minimum OCAP requirement for standard-definition graphic devices resolution is 640x480. The minimum OCAP requirement for high-definition devices is 960x540, in addition to 640x480.

The graphics plane is typically limited in the number of colors or in the size of the plane.

Video

The video plane is positioned in front of the background plane. Scaling and position of the video plane is limited based on the target environment. Scaling is typically limited to full-screen or quarter-screen. Positioning of the video plane is typically limited to certain areas on the screen.

The minimum OCAP requirement for standard-definition input video resolution is 720x480 or 640x480. The minimum OCAP requirement for high-definition input video resolution is 1920x1080.

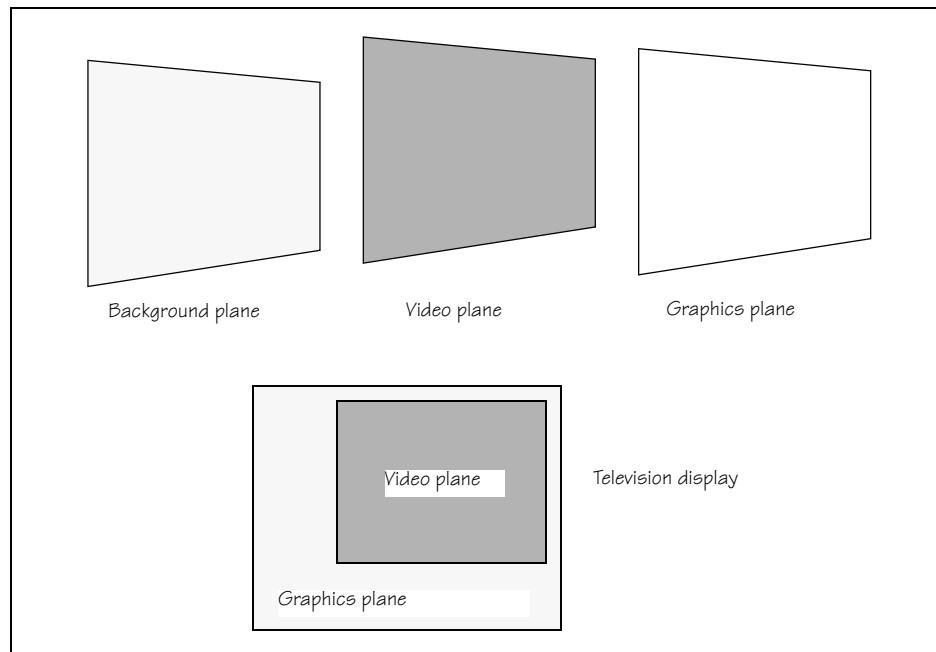


FIGURE DSP-1:

Planes and television display

Coordinate system

The graphics coordinate system is anchored in the upper left-hand corner of the screen, with coordinates increasing down and to the right. Coordinates lie between pixels of the screen. Operations that draw outlines of shapes traverse a coordinate path with a pen that hangs beneath and to the right of the path. The size of the pen is always one pixel in width and height.

◆ **NOTE:** Information in this section is partly taken from [Graphic Java: Mastering the JFC Volume I: AWT](#).

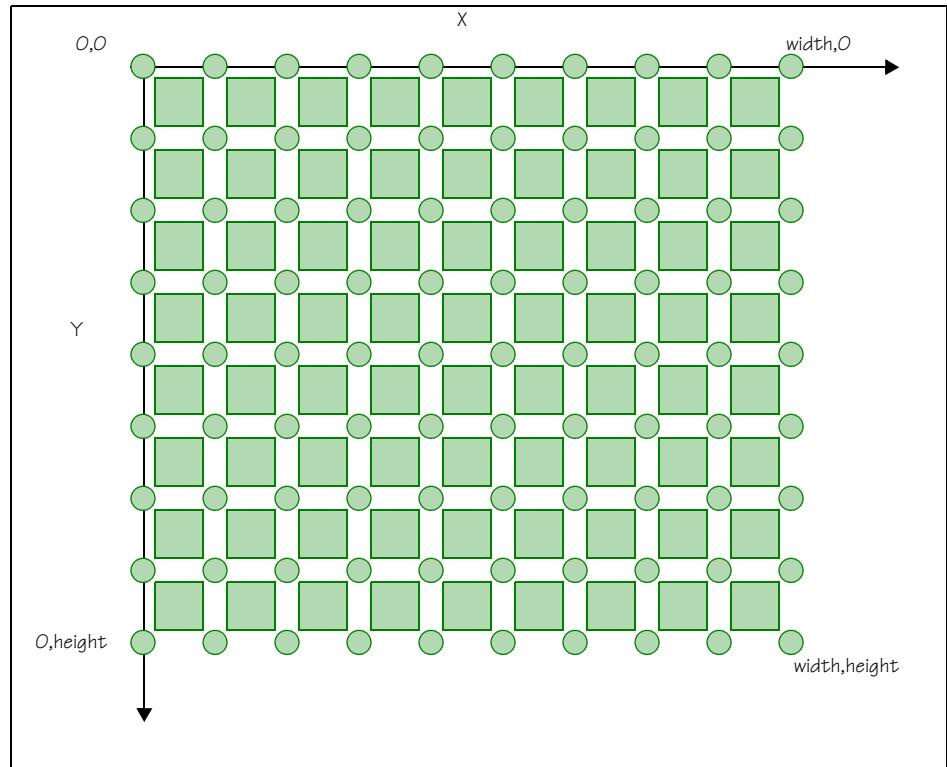


FIGURE DSP-2:

Graphics coordinate system

It is perfectly valid to refer to a location using a negative coordinate. Negative coordinates indicate an imaginary point off of the screen (for screen coordinates) or a point relative to the translated origin. In some cases (for example, `mpeos_gfxBitBlt` and `mpeos_gfxStretchBlt`), it is also valid to use a negative width or height to indicate a translation about one or both axis.

Surface

A surface corresponds to the graphics plane of the physical screen or an off-screen pixel map. Rendering operations are performed on a drawing surface through a specific context. Synonyms include bitmap, pixel map (or pixmap), or raster.

Graphic context

A graphic context provides the context through which rendering operations take place. A graphic context defines and manages the attributes that affect rendering.

Fonts

Fonts define the glyphs used to render text on the screen. OCAP requires support of PFR fonts but the actual support for specific font formats is within the MPEOS implementation for a given platform.

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the display functions:

`MPE_EINVAL`

`MPE_SUCCESS`

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

`MPE_EINVAL`

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

`MPE_SUCCESS`

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h`, `mpe_types.h`, `mpeos_display.h`, and `mpeos_gfx.h`, are used by the display functions:

| | |
|---|--|
| <code>mpe_Bool</code> | <code>mpe_Displ1394DeviceInfo</code> |
| <code>mpe_Displ1394Devices</code> | <code>mpe_DisplBGImage</code> |
| <code>mpe_DisplCoherentConfig</code> | <code>mpe_DisplDevice</code> |
| <code>mpe_DisplDeviceConfig</code> | <code>mpe_DisplDeviceConfigInfo</code> |
| <code>mpe_DisplDeviceInfo</code> | <code>mpe_DisplDeviceType</code> |
| <code>mpe_DisplDfcAction</code> | <code>mpe_DisplError</code> |
| <code>mpe_DisplOutputPort</code> | <code>mpe_DisplOutputPortInfo</code> |
| <code>mpe_DisplOutputPortOption</code> | <code>mpe_DisplOutputPortOptionName</code> |
| <code>mpe_DisplOutputPortOptionValue</code> | <code>mpe_DisplOutputPortType</code> |
| <code>mpe_DisplScreen</code> | <code>mpe_DisplScreenArea</code> |
| <code>mpe_DisplScreenInfo</code> | <code>mpe_Error</code> |
| <code>mpe_GfxColor</code> | <code>mpe_GfxDimensions</code> |
| <code>mpe_GfxRectangle</code> | <code>mpe_GfxSurface</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Displ1394DeviceInfo`

`mpe_Displ1394DeviceInfo` specifies the structure for describing a specific IEEE-1394 device. `mpe_Displ1394DeviceInfo` is defined in `mpeos_display.h` as follows:

```
typedef struct {
    int id;
    char eui64[8];
    char vendor[128];
    char model[128];
    int subunitType;
} mpe_Displ1394DeviceInfo;
```

where

| | |
|---------------------|--|
| <code>id</code> | specifies a platform-specific identifier for the attached device. |
| <code>eui64</code> | indicates the value of EUI-64 of the 1394 node. EUI-64 is defined in IEEE Standard 1394-1995. |
| <code>vendor</code> | indicates the value of VENDOR NAME TEXTUAL DESCRIPTOR of the 1394 node. If the 1394 node does not have the VENDOR NAME TEXTUAL DESCRIPTOR, null is returned. |

model indicates the value of MODEL NAME TEXTUAL DESCRIPTOR of the 1394 node. MODEL NAME TEXTUAL DESCRIPTOR is defined in EIA-775-A.

subunitType indicates the list of subunitTypes supported by the 1394 node. The subunit type is defined in EIA-775-A.

mpe_Displ1394Devices

mpe_Displ1394Devices specifies the structure used to define the IEEE-1394 device information array. **mpe_Displ1394Devices** is used by the **mpeos_dispGetVideoOutputPortOption()** and **mpeos_dispSetVideoOutputPortOption()**. **mpe_Displ1394Devices** is defined in **mpeos_display.h** as follows:

```
typedef struct {
    int numEntry;
    mpe_Displ1394DeviceInfo pInfo[1];
} mpe_Displ1394Devices;
```

where

numEntry indicates the number of entries provided in the **pInfo** array.

pInfo indicates an array of **mpe_Displ1394DeviceInfo** entries identifying the devices connected to the IEEE-1394 port. **mpe_Displ1394DeviceInfo** is described in the **mpe_Displ1394DeviceInfo** section.

mpe_DispBGImage

mpe_DispBGImage is a pointer to the background image handle. It serves as an abstraction for the background image. **mpe_DispBGImage** is defined in **mpeos_display.h** as follows:

```
typedef struct { int unused1; } * mpe_DispBGImage;
```

mpe_DispCoherentConfig

mpe_DispCoherentConfig is a pointer to the screen device coherent configurations handle. It serves as an abstraction for a coherent set of configurations for multiple devices. A coherent set of configurations is a set of configurations that can be used together. **mpe_DispCoherentConfig** is defined in **mpeos_display.h** as follows:

```
typedef struct { int unused1; } * mpe_DispCoherentConfig;
```

mpe_DispDevice

mpe_DispDevice is a pointer to a device handle. It serves as an abstraction for a screen device belonging to a single screen. The screen is defined as video, graphics, or background type. **mpe_DispDevice** is defined in **mpeos_display.h** as follows:

```
typedef struct { int unused1; } * mpe_DispDevice;
```

mpe_DispDeviceConfig

`mpe_DispDeviceConfig` is a pointer to the device configuration handle. It serves as an abstraction for a discrete configuration for a screen device. `mpe_DispDeviceConfig` is defined in `mpeos_display.h` as follows:

```
typedef struct { int unused1; } * mpe_DispDeviceConfig;
```

mpe_DispDeviceConfigInfo

`mpe_DispDeviceConfigInfo` specifies the structure which provides fields for describing a device configuration. If the associated configuration is the non-contributing configuration, then all values are expected to be zero.

`mpe_DispDeviceConfigInfo` is defined in `mpeos_display.h` as follows:

```
typedef struct {
    mpe_DispDevice device;
    mpe_Bool flickerFilter;
    mpe_Bool interlaced;
    mpe_GfxDimensions resolution;
    mpe_GfxDimensions pixelAspectRatio;
    mpe_DispScreenArea area;
    mpe_Bool mpegStills;
    mpe_Bool changeableColor;
} mpe_DispDeviceConfigInfo;
```

where

device indicates the device to which this configuration belongs. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

flickerFilter determines if the flicker filter is supported. If `flickerFilter` is TRUE, it indicates the flicker filter is supported. If `flickerFilter` is FALSE, it indicates the flicker filter is not supported. `mpe_Bool` is described in the *mpe_Bool* section.

interlaced determines if the configuration is interlaced. If `interlaced` is TRUE, the configuration is interlaced. If `interlaced` is FALSE, the configuration is progressive-scan (not interlaced). `mpe_Bool` is described in the *mpe_Bool* section.

resolution indicates the pixel resolution of this configuration. If this configuration is the video non-contributing configuration, it will be {0,0}. `mpe_GfxDimensions` is described in the *mpe_GfxDimensions* section.

pixelAspectRatio indicates the pixel aspect ratio of this configuration. `mpe_GfxDimensions` is described in the *mpe_GfxDimensions* section.

area indicates the normalized screen area covered by this configuration. `mpe_DispScreenArea` is described in the *mpe_DispScreenArea* section.

`mpegStills` specifies background device configurations only. If `mpegStills` is TRUE, it indicates the MPEG I-frame stills are supported by this configuration. If `mpegStills` is FALSE, it indicates the MPEG I-frame stills are not supported by this configuration. `mpe_Bool` is described in the *mpe_Bool* section.

`changeableColor`

specifies background device configurations only. If `changeableColor` is TRUE, a changeable single color is supported by this configuration. If `changeableColor` is FALSE, a changeable single color is not supported by this configuration. `mpe_Bool` is described in the *mpe_Bool* section.

`mpe_DispDeviceInfo`

`mpe_DispDeviceInfo` specifies the structure that provides fields for describing a device. `mpe_DispDeviceInfo` is defined in `mpeos_display.h` as follows:

```
typedef struct {
    mpe_DispDeviceType type;
    const char * idString;
    mpe_DispScreen screen;
    mpe_GfxDimensions screenAspectRatio;
} mpe_DispDeviceInfo;
```

where

`type` indicates the type of device. `mpe_DispDeviceType` is described in the *mpe_DispDeviceType* section. Currently, the following values are supported for type:

`MPE_DISPLAY_ALL_DEVICES`

gets the total count of all device types for the screen.

`MPE_DISPLAY_GRAPHICS_DEVICE`

indicates a graphic device type.

`MPE_DISPLAY_VIDEO_DEVICE`

indicates a video device.

`MPE_DISPLAY_BACKGROUND_DEVICE`

indicates a background device.

`idString` is a pointer to the identification string associated with the device.

`screen` indicates the parent screen for the device. `mpe_DispScreen` is described in the *mpe_DispScreen* section.

`screenAspectRatio`

indicates the screen aspect ratio for the device.

`mpe_GfxDimensions` is described in the *mpe_GfxDimensions* section.

mpe_DispDeviceType

`mpe_DispDeviceType` specifies the enumeration describing the type of device (`mpe_DispDevice`). `mpe_DispDeviceType` is defined in `mpeos_display.h` as follows:

```
typedef enum {
    MPE_DISPLAY_ALL_DEVICES = 0,
    MPE_DISPLAY_GRAPHICS_DEVICE,
    MPE_DISPLAY_VIDEO_DEVICE,
    MPE_DISPLAY_BACKGROUND_DEVICE,
} mpe_DispDeviceType;
```

where

| | |
|--|--|
| <code>MPE_DISPLAY_ALL_DEVICES</code> = 0 | gets input for <code>mpeos_dispGetDeviceCount()</code> and <code>mpeos_dispGetDevices()</code> of the total count of all device types and to retrieve all devices. Otherwise, not a valid device type. |
| <code>MPE_DISPLAY_GRAPHICS_DEVICE</code> | indicates a graphic device type. |
| <code>MPE_DISPLAY_VIDEO_DEVICE</code> | indicates a video device. |
| <code>MPE_DISPLAY_BACKGROUND_DEVICE</code> | indicates a background device. |

mpe_DispDfcAction

`mpe_DispDfcAction` specifies the Decoder Format Conversion (DFC) constants. `mpe_DispDfcAction` is defined in `mpeos_display.h` as follows:

```
typedef enum {
    MPE_DFC_PROCESSING_NONE = 0,
    MPE_DFC_PROCESSING_FULL,
    MPE_DFC_PROCESSING_LB_16_9,
    MPE_DFC_PROCESSING_LB_14_9,
    MPE_DFC_PROCESSING_CCO,
    MPE_DFC_PROCESSING_PAN_SCAN,
    MPE_DFC_PROCESSING_LB_2_21_1_ON_4_3,
    MPE_DFC_PROCESSING_LB_2_21_1_ON_16_9,
    MPE_DFC_PLATFORM,
    MPE_DFC_PROCESSING_16_9_ZOOM
} mpe_DispDfcAction;
```

where

| | |
|---|---|
| <code>MPE_DFC_PROCESSING_NONE</code> | indicates the decoder format conversion is inactive. |
| <code>MPE_DFC_PROCESSING_FULL</code> | indicates the full 720x576 frame is transferred (this may be either 4:3 or 16:9; part of this may be black (for example, in pillar box cases)). |
| <code>MPE_DFC_PROCESSING_LB_16_9</code> | indicates the 720x576 input grid is transferred into a 16:9 letterbox format in a 4:3 frame. |

MPE_DFC_PROCESSING_LB_14_9
 indicates the 720x576 input grid is transferred into a 14:9 letterbox format in a 4:3 frame.

MPE_DFC_PROCESSING_CCO
 indicates the 4:3 central part of an 720x576 input 16:9 frame is transferred into a 720x576 4:3 output frame.

MPE_DFC_PROCESSING_PAN_SCAN
 indicates no decoder format conversion.

MPE_DFC_PROCESSING_LB_2_21_1_ON_4_3
 indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 4:3 frame.

MPE_DFC_PROCESSING_LB_2_21_1_ON_16_9
 indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 16:9 frame.

MPE_DFC_PLATFORM
 indicates the decoder format conversion for the platform is used. Setting DFC to MPE_DFC_PLATFORM returns control of the format conversion to the platform, allowing the platform to apply any DFC it feels appropriate. Once set, this is not returned as the current DFC. Instead, the actual DFC in use by the platform at the time is returned.

MPE_DFC_PROCESSING_16_9_ZOOM
 indicates the central 16:9 letterbox area of the 4:3 720x576 input grid is expanded to fill the 16:9 output frame.

mpe_DispError

`mpe_DispError` specifies the error codes. `mpe_DispError` is defined in `mpeos_display.h` as follows:

```
typedef enum mpe_DispError {
    MPE_DISP_ERROR_NO_ERROR = MPE_SUCCESS,
    MPE_DISP_ERROR_FIRST_ERROR = 0,
    MPE_DISP_ERROR_IMPACTS_OTHERS,
    MPE_DISP_ERROR_UNKNOWN,
    MPE_DISP_ERROR_NO_MEMORY,
    MPE_DISP_ERROR_INVALID_PARAM,
    MPE_DISP_ERROR_UNIMPLEMENTED,
    MPE_DISP_ERROR_BAD_IFRAME,
    MPE_DISP_ERROR_BGCOLOR,
} mpe_DispError;
```

where

MPE_DISP_ERROR_NO_ERROR
 specifies the call was successful.

MPE_DISP_ERROR_FIRST_ERROR
 modifies to be the base of display errors.

MPE_DISP_ERROR_IMPACTS_OTHERS
 indicates the configuration setting may implicitly affect other devices.

MPE_DISP_ERROR_UNKNOWN
 indicates a generic error.

MPE_DISP_ERROR_NO_MEMORY
 indicates the memory required could not be allocated.

MPE_DISP_ERROR_INVALID_PARAM
 indicates an invalid parameter was passed to the function.

MPE_DISP_ERROR_UNIMPLEMENTED
 indicates unimplemented support.

MPE_DISP_ERROR_BAD_IFRAME
 indicates the I-frame was rejected.

MPE_DISP_ERROR_BGCOLOR
 indicates a background setting problem was encountered.

mpe_DispatchOutputPort

`mpe_DispatchOutputPort` specifies the display port handle.
`mpe_DispatchOutputPort` is defined in `mpeos_display.h` as follows:

```
typedef struct { int unused1; } *mpe_DispatchOutputPort;
```

mpe_DispatchOutputPortInfo

`mpe_DispatchOutputPortInfo` specifies the structure which provides fields for describing an output port. `mpe_DispatchOutputPortInfo` is defined in `mpeos_display.h` as follows:

```
typedef struct {
    mpe_DispatchOutputPortType type;
    mpe_Bool enabled;
    mpe_Bool dtcpSupported;
    mpe_Bool hdcpSupported;
    int32 restrictedResolution;
} mpe_DispatchOutputPortInfo;
```

where

`type` indicates the type of the port. `mpe_DispatchOutputPortType` is described in the *mpe_DispatchOutputPortType* section.
 Currently, the following values are supported for type:

MPE_DISPLAY_RF_PORT
 indicates an RF port.

MPE_DISPLAY_BASEBAND_PORT
 indicates a baseband port.

MPE_DISPLAY_SVIDEO_PORT
 indicates an S-Video port.

| | |
|----------------------|--|
| | MPE_DISPLAY_1394_PORT indicates a 1394 port. |
| | MPE_DISPLAY_DVI_PORT indicates a DVI port. |
| | MPE_DISPLAY_COMPONENT_PORT indicates a component port. |
| enabled | determines whether the output port is enabled. If enabled is TRUE, the port was enabled at the time <code>mpeos_dispGetOutputPortInfo()</code> was called. If enabled is FALSE, the port was not enabled at the time <code>mpeos_dispGetOutputPortInfo()</code> was called. <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |
| dtcpSupported | determines whether DTCP is supported on the port. If dtcpSupported is TRUE, the port supports DTCP. If dtcpSupported is FALSE, the port does not support DTCP. <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |
| hdcpSupported | determines whether HDCP is supported on the port. If hdcpSupported is TRUE, the port supports HDCP. If hdcpSupported is FALSE, the port does not support HDCP. <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |
| restrictedResolution | indicates the restricted vertical resolution of high-definition output for the port. If there is no restriction, the maximum vertical resolution is specified. |

mpe_DispatchOutputPortOption

`mpe_DispatchOutputPortOption` sets and gets an option and value for the display. `mpe_DispatchOutputPortOption` is used when calling `mpeos_dispGetVideoOutputPortOption()` and `mpeos_dispSetVideoOutputPortOption()`. `mpe_DispatchOutputPortOption` is defined in `mpeos_display.h` as follows:

```
typedef struct {
    mpe_DispatchOutputPortOptionName option;
    void *value;
} mpe_DispatchOutputPortOption;
```

where

| | |
|--------|--|
| option | sets or gets the name of the option. <code>mpe_DispatchOutputPortOptionName</code> is described in the <i>mpe_DispatchOutputPortOptionName</i> section. Currently, the following values are supported for option: |
|--------|--|

MPE_DISP_1394_SELECT_JACK

sets or gets the analog or digital type of video input port for the display.

MPE_DISP_1394_DEVICE_LIST

sets or gets a list of all the devices connected to the set-top box (for example, storage device or video recorder).

MPE_DISP_1394_MODEL_NAME

sets or gets the model name for the current display device.

MPE_DISP_1394_VENDOR_NAME

sets or gets the vendor name for the current display device.

MPE_DISP_1394_SUBUNIT_TYPE

sets or gets the subunit type for the current display device.

MPE_DISP_1394_SELECT_SINK

sets or gets the sink node for stream connection to the current display.

value

is an input or output pointer to the value for *option*.

Currently, the following values are supported for **value**:

MPE_DISP_1394_SELECT_JACK_TYPE_ANALOG

specifies an analog jack. May be used if *option* is **MPE_DISP_1394_SELECT_JACK**.

MPE_DISP_1394_SELECT_JACK_TYPE_DIGITAL

specifies a digital jack. May be used if *option* is **MPE_DISP_1394_SELECT_JACK**.

mpe_DispatchPortOptionName

mpe_DispatchPortOptionName specifies the video port options that can be returned and set with **mpe_DispatchPortOption**.

mpe_DispatchPortOptionName is defined in **mpeos_display.h** as follows:

```
typedef enum {
    MPE_DISP_1394_SELECT_JACK,
    MPE_DISP_1394_DEVICE_LIST,
    MPE_DISP_1394_MODEL_NAME,
    MPE_DISP_1394_VENDOR_NAME,
    MPE_DISP_1394_SUBUNIT_TYPE
    MPE_DISP_1394_SELECT_SINK
} mpe_DispatchPortOptionName;
```

where

MPE_DISP_1394_SELECT_JACK

specifies the video input port for the display.

MPE_DISP_1394_DEVICE_LIST

specifies a list of all the devices connected to the set-top box (for example, storage device or video recorder).

MPE_DISP_1394_MODEL_NAME

specifies the model name for the current display device.

MPE_DISP_1394_VENDOR_NAME
specifies the vendor name for the current display device.

MPE_DISP_1394_SUBUNIT_TYPE
specifies the subunit type for the current display device.

MPE_DISP_1394_SELECT_SINK
sets or gets the sink node for stream connection to the current display. Verify the description.

mpe_DispatchOutputPortOptionValue

`mpe_DispatchOutputPortOptionValue` identifies a command to send to a display device (for example, Digital TV) connected to the IEEE-1394 port. `mpe_DispatchOutputPortOptionValue` is used by the OCAP stack to ensure that EAS graphics are properly displayed (as digital output on the IEEE-1394 port is not expected to include graphics overlay).

`mpe_DispatchOutputPortOptionValue` is used by the `mpeos_dispSetVideoOutputPortOption()`.

`mpe_DispatchOutputPortOptionValue` is defined in `mpeos_display.h` as follows:

```
typedef enum {
    MPE_DISP_1394_SELECT_JACK_TYPE_ANALOG,
    MPE_DISP_1394_SELECT_JACK_TYPE_DIGITAL
} mpe_DispatchOutputPortOptionValue;
```

where

MPE_DISP_1394_SELECT_JACK_TYPE_ANALOG

specifies the command to switch the display to analog input.

MPE_DISP_1394_SELECT_JACK_TYPE_DIGITAL

specifies the command to switch the display to digital input.

mpe_DispatchOutputPortType

`mpe_DispatchOutputPortType` specifies an enumeration describing the type of output port. `mpe_DispatchOutputPortType` is defined in `mpeos_display.h` as follows:

```
typedef enum {
    MPE_DISPLAY_RF_PORT = 0,
    MPE_DISPLAY_BASEBAND_PORT,
    MPE_DISPLAY_SVIDEO_PORT,
    MPE_DISPLAY_1394_PORT,
    MPE_DISPLAY_DVI_PORT,
    MPE_DISPLAY_COMPONENT_PORT
} mpe_DispatchOutputPortType;
```

where

MPE_DISPLAY_RF_PORT

indicates an RF port.

MPE_DISPLAY_BASEBAND_PORT

indicates a baseband port.

MPE_DISPLAY_SVIDEO_PORT
indicates a S-Video port.

MPE_DISPLAY_1394_PORT
indicates a 1394 port.

MPE_DISPLAY_DVI_PORT
indicates a DVI port.

MPE_DISPLAY_COMPONENT_PORT
indicates a component port.

mpe_DispScreen

mpe_DispScreen is a pointer to the screen handle. It serves as an abstraction for an actual display screen, that supports one or more screen devices. Each screen is defined as video, graphics, or background type. **mpe_DispScreen** is defined in **mpeos_display.h** as follows:

```
typedef struct { int unused1; } * mpe_DispScreen;
```

mpe_DispScreenArea

mpe_DispScreenArea specifies the normalized screen coordinates area. **mpe_DispScreenArea** is defined in **mpeos_display.h** as follows:

```
typedef struct {
    float x;
    float y;
    float width;
    float height;
} mpe_DispScreenArea;
```

where

x indicates the vertical screen position.

y indicates the horizontal screen position.

width indicates the width of the screen.

height indicates the height of the screen.

mpe_DispScreenInfo

`mpe_DispScreenInfo` identifies the presence of a non-contributing video configuration. A non-contributing video configuration indicates the background display device actually shares hardware resources with the video display device. When the background device is displaying a still image, the video device must be in a non-contributing video configuration. It is expected that the need for a non-contributing configuration is global to all video devices. `mpe_DispScreenInfo` is defined in `mpeos_display.h` as follows:

```
typedef struct {
    mpe_DispDeviceConfig notContributing;
} mpe_DispScreenInfo;
```

where

notContributing

indicates whether there is a non-contributing video configuration or not. If the value returned is non-zero, the value is the `mpe_DispDeviceConfig` handle for the non-contributing video configuration. If the value returned is 0, there is no video non-contributing configuration. `mpe_DispDeviceConfig` is described in the *mpe_DispDeviceConfig* section.

mpe_Error

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

mpe_GfxColor

`mpe_GfxColor` specifies a 32-bit color value representing alpha, red, green, and blue quadruplet. `mpe_GfxColor` is described in `mpeos_gfx.h` as follows:

```
typedef uint32_t mpe_GfxColor;
```

mpe_GfxDimensions

`mpe_GfxDimensions` specifies the dimensions of the graphics element. `mpe_GfxDimensions` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxDimensions {
    int32 width;
    int32 height;
} mpe_GfxDimensions;
```

where

`width` indicates the width of the graphic element.

`height` indicates the height of the graphic element.

mpe_GfxRectangle

`mpe_GfxRectangle` defines the origin and dimensions of a rectangle.
`mpe_GfxRectangle` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxRectangle {  
    int32 x;  
    int32 y;  
    int32 width;  
    int32 height;  
} mpe_GfxRectangle;
```

where

- x indicates the x coordinate to use as the bottom-left corner of the rectangle.
- y indicates the y coordinate to use as the bottom-left corner of the rectangle.
- width indicates the width of the rectangle.
- height indicates the height of the rectangle.

mpe_GfxSurface

`mpe_GfxSurface` is a handle to a graphics surface. `mpe_GfxSurface` is defined in `mpeos_gfx.h` as follows:

```
typedef struct _mpeH_GfxSurface_t {  
    int unused1;  
} *mpe_GfxSurface;
```

Events

The following display events, which are defined in `mpeos_disp.h`, are used by the display functions:

`MPE_DFC_CHANGED`

`MPE_DFC_CHANGED` is sent to the decoder queue if the implementation changes the current Decoder Format Conversion (DFC).

`MPE_DFC_CHANGED` is not optional for implementation that can control the DFC filter. `MPE_DFC_CHANGED` is defined as follows:

```
#define MPE_DFC_CHANGED 0x17
```

Supported functions

The following display functions need to be supported:

- `mpeos_dispBGImageDelete`
deletes a background image.
- `mpeos_dispBGImageGetSize`
determines the size of the image.
- `mpeos_dispBGImageNew`
creates a new background image.
- `mpeos_dispCheckDFC`
verifies DFC is supported.
- `mpeos_dispDisplayBGImage`
displays a background image.
- `mpeos_dispEnableOutputPort`
enables or disables the output port.
- `mpeos_dispFlushGfxSurface`
flushes the main surface buffer for the given graphics device.
- `mpeos_dispGetBGColor`
gets the background color.
- `mpeos_dispGetCoherentConfigCount`
gets the number of coherent screen configurations.
- `mpeos_dispGetCoherentConfigs`
gets the screen configurations.
- `mpeos_dispGetConfigCount`
gets the number of device configurations.
- `mpeos_dispGetConfigInfo`
gets configuration information.
- `mpeos_dispGetConfigs`
gets the device configurations.
- `mpeos_dispGetConfigSet`
gets a set of configurations.
- `mpeos_dispGetConfigSetCount`
gets the number of device configurations.
- `mpeos_dispGetCurrConfig`
gets the current device configuration.
- `mpeos_dispGetDeviceCount`
gets the number of supported devices.
- `mpeos_dispGetDeviceInfo`
gets device information.
- `mpeos_dispGetDevices`
gets the devices of a given type for the screen.

`mpeos_dispGetDFC`
gets the current DFC for the specified decoder.

`mpeos_dispGetGfxSurface`
gets a handle for the screen surface for the specified device.

`mpeos_dispGetOutputPortCount`
gets the number of video output ports.

`mpeos_dispGetOutputPortInfo`
gets output port information.

`mpeos_dispGetOutputPorts`
gets the video output ports.

`mpeos_dispGetRFBypassState`
gets the state of the RF bypass.

`mpeos_dispGetRFChannel`
gets the RF output channel.

`mpeos_dispGetScreenCount`
gets a count of all supported the display screens.

`mpeos_dispGetScreenInfo`
gets screen information.

`mpeos_dispGetScreens`
gets all the screen handles.

`mpeos_dispGetVideoOutputPortOption`
gets the value for a display option.

`mpeos_dispSetBGColor`
sets the background color.

`mpeos_dispSetCoherentConfig`
sets the screen configurations.

`mpeos_dispSetCurrConfig`
sets the device configuration.

`mpeos_dispSetDFC`
sets the DFC for the decoder.

`mpeos_dispSetRFBypassState`
sets the RF bypass state.

`mpeos_dispSetRFChannel`
sets the RF output channel.

`mpeos_dispSetRFChannel`
sets the RF output channel.

`mpeos_dispSetVideoOutputPortOption`
sets the option on a video output port

`mpeos_dispWouldImpact`
identifies the impact on other devices.

mpeos_dispBGImageDelete

deletes a background image.

syntax mpe_Error mpeos_dispBGImageDelete(mpe_DispBGImage *image*);

parameter(s) *image* specifies the background image to delete. *image* is returned by a previous call to mpeos_dispBGImageNew().
mpe_DispBGImage is described in the *mpe_DispBGImage* section.

value returned If the call is successful, mpeos_dispBGImageDelete() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_DISP_ERROR_UNIMPLEMENTED
indicates unimplemented support.

description mpeos_dispBGImageDelete() should delete the specified image, freeing up any previously allocated resources.

related function(s) mpeos_dispBGImageGetSize
mpeos_dispBGImageNew
mpeos_dispDisplayBGImage
mpeos_dispGetBGCColor
mpeos_dispSetBGCColor

mpeos_dispBGImageGetSize

determines the size of the image.

syntax `mpe_Error mpeos_dispBGImageGetSize(mpe_DispBGImage image, mpe_GfxDimensions * size);`

parameter(s) *image* specifies the image. *image* is returned by a previous call to `mpeos_dispBGImageNew()`. `mpe_DispBGImage` is described in the *mpe_DispBGImage* section.

size is an output pointer to the dimensions of the graphics element. `mpe_GfxDimensions` is described in the *mpe_GfxDimensions* section.

value returned If the call is successful, `mpeos_dispBGImageGetSize()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispBGImageGetSize()` should determine the size of the Moving Picture Experts Group - broadcast signals (MPEG-2) I-frame image.

related function(s) `mpeos_dispBGImageDelete`
`mpeos_dispBGImageNew`
`mpeos_dispDisplayBGImage`
`mpeos_dispGetBGColor`
`mpeos_dispSetBGColor`

mpeos_dispBGImageNew

creates a new background image.

syntax mpe_Error mpeos_dispBGImageNew(
 uint8_t * *buffer*,
 size_t *length*,
 mpe_DispBGImage * *image*);

parameter(s) *buffer* is an input pointer to the byte array containing the MPEG-2 I-frame.

length specifies the number of bytes in the byte array.

image is an output pointer to the new background image.
mpe_DispBGImage is described in the *mpe_DispBGImage* section.

value returned If the call is successful, *mpeos_dispBGImageNew()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE_DISP_ERROR_INVALID_PARAM

indicates an invalid parameter was passed to the function.

description *mpeos_dispBGImageNew()* should create a new background image. If background image support is provided by a software decoder, the MPEG-2 I-frame should be decoded or fail if there are errors. The image is decoded only once in software.

related function(s) *mpeos_dispBGImageDelete*
mpeos_dispBGImageGetSize
mpeos_dispDisplayBGImage
mpeos_dispGetBGColor
mpeos_dispSetBGColor

mpeos_dispCheckDFC

verifies DFC is supported.

syntax mpe_Error mpeos_dispCheckDFC(
 mpe_DispDevice *decoder*,
 mpe_DispDfcAction *action*);

| | | |
|---------------------|--------------------------------------|---|
| parameter(s) | <i>decoder</i> | specifies the decoder. <i>mpe_DispDevice</i> is described in the <i>mpe_DispDevice</i> section. |
| | <i>action</i> | specifies the action is a DFC. <i>mpe_DispDfcAction</i> is described in the <i>mpe_DispDfcAction</i> section. Currently, the following values are supported for <i>action</i> : |
| | MPE_DFC_PROCESSING_NONE | indicates the decoder format conversion is inactive. |
| | MPE_DFC_PROCESSING_FULL | indicates the full 720x576 frame is transferred (this may be either 4:3 or 16:9; part of this may be black (for example, in pillar box cases)). |
| | MPE_DFC_PROCESSING_LB_16_9 | indicates the 720x576 input grid is transferred into a 16:9 letterbox format in a 4:3 frame. |
| | MPE_DFC_PROCESSING_LB_14_9 | indicates the 720x576 input grid is transferred into a 14:9 letterbox format in a 4:3 frame. |
| | MPE_DFC_PROCESSING_CCO | indicates the 4:3 central part of an 720x576 input 16:9 frame is transferred into a 720x576 4:3 output frame. |
| | MPE_DFC_PROCESSING_PAN_SCAN | indicates no decoder format conversion. |
| | MPE_DFC_PROCESSING_LB_2_21_1_ON_4_3 | indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 4:3 frame. |
| | MPE_DFC_PROCESSING_LB_2_21_1_ON_16_9 | indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 16:9 frame. |
| | MPE_DFC_PLATFORM | indicates the decoder format conversion for the platform is used. |
| | MPE_DFC_PROCESSING_16_9_ZOOM | indicates the central 16:9 letterbox area of the 4:3 720x576 input grid is expanded to fill the 16:9 output frame. |

value returned If the call is successful, `mpeos_dispCheckDFC()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE_ERROR_MEDIA_OS`
specifies an operating system error.

description `mpeos_dispCheckDFC()` should verifies DFC is supported.

related function(s) `mpeos_dispGetDFC`
`mpeos_dispSetDFC`

mpeos_dispDisplayBGImage

displays a background image.

syntax

```
mpe_Error mpeos_dispDisplayBGImage(
    mpe_DispDevice device,
    mpe_DispBGImage image,
    mpe_GfxRectangle * area );
```

| | | |
|---------------------|---------------|---|
| parameter(s) | <i>device</i> | specifies the device to display the <i>image</i> . <i>device</i> must be an MPE_DISPLAY_BACKGROUND_DEVICE type that was returned by a previous call to <code>mpeos_dispGetDevices()</code> . <code>mpe_DispDevice</code> is described in the <i>mpe_DispDevice</i> section. |
| | <i>image</i> | specifies the handle for the MPEG I-frame image. <i>image</i> is returned by a previous call to <code>mpeos_dispBGImageNew()</code> . <code>mpe_DispBGImage</code> is described in the <i>mpe_DispBGImage</i> section. |
| | <i>area</i> | is an input pointer to the location on the screen to display the <i>image</i> . <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |

value returned If the call is successful, `mpeos_dispDisplayBGImage()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_DISP_ERROR_INVALID_PARAM`

indicates an invalid parameter was passed to the function.

`MPE_DISP_ERROR_UNIMPLEMENTED`

indicates unimplemented support.

description `mpeos_dispDisplayBGImage()` should instruct the background device to display the MPEG I-frame image. The contents of the MPEG I-frame buffer should be considered to be copied out of the passed-in object, allowing the caller to free the object on return.

The implementation determines whether to crop, tile, or scale the image. The positioning of the image is also implementation dependent. The current background color should be used wherever the image is not displayed.

related function(s)

- `mpeos_dispBGImageDelete`
- `mpeos_dispBGImageGetSize`
- `mpeos_dispBGImageNew`
- `mpeos_dispGetBGColor`
- `mpeos_dispSetBGColor`

mpeos_dispEnableOutputPort

enables or disables the output port.

syntax mpe_Error mpeos_dispEnableOutputPort(
 mpe_DispOutputPort *port*,
 mpe_Bool *enable*);

parameter(s) *port* specifies the output port. *port* is returned from a previous call to mpeos_dispGetOutputPorts. mpe_DispOutputPort is described in the *mpe_DispOutputPort* section.

enable determines whether or not the port is enabled. If *enable* is TRUE, the port is enabled. If *enable* is FALSE, the port is disabled. mpe_Bool is described in the *mpe_Bool* section.

value returned If the call is successful, mpeos_dispEnableOutputPort() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_DISP_ERROR_INVALID_PARAM
indicates an invalid parameter was passed to the function.

description mpeos_dispEnableOutputPort() should enable or disable the output port, based on the value specified by the *enable* parameter.

related function(s) mpeos_dispGetOutputPortInfo
mpeos_dispGetOutputPortInfo
mpeos_dispGetOutputPorts

mpeos_dispFlushGfxSurface

flushes the main surface buffer for the given graphics device.

syntax mpe_Error mpeos_dispFlushGfxSurface(mpe_DispDevice device);

parameter(s) *device* specifies the device to flush. *mpe_DispDevice* is described in the *mpe_DispDevice* section.

value returned If the call is successful, *mpeos_dispFlushGfxSurface()* should return *MPE_GFX_ERROR_NOERR*. Otherwise, it should return one of the following error codes:

MPE_GFX_ERROR_OSERR
indicates an error in the OS layer.

MPE_GFX_ERROR_UNIMPLEMENTED
indicates the feature has not been implemented.

description *mpeos_dispFlushGfxSurface()* should flush the contents of the main surface buffer for the given graphics device, if the surface is buffered. If the device does not buffer surfaces, or if flushes are implicit in each drawing operation, then *mpeos_dispFlushGfxSurface()* does nothing.

related function(s) *mpeos_dispGetGfxSurface*

mpeos_dispGetBGColor

gets the background color.

syntax mpe_Error mpeos_dispGetBGColor(
 mpe_DispDevice *device*,
 mpe_GfxColor * *color*);

parameter(s) *device* specifies the device to display the *color*. *device* must be an MPE_DISPLAY_BACKGROUND_DEVICE type that was returned by a previous call to mpeos_dispGetDevices(). *mpe_DispDevice* is described in the *mpe_DispDevice* section.

color is an output pointer to the color. *mpe_GfxColor* is described in the *mpe_GfxColor* section.

value returned If the call is successful, mpeos_dispGetBGColor() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_DISP_ERROR_UNIMPLEMENTED
indicates unimplemented support.

description mpeos_dispGetBGColor() should get the current background color associated with the background device. The value returned is not guaranteed to be the color set on the last call to mpeos_dispSetBGColor(); it may reflect a reduced-color palette used by the implementation. If the device is currently displaying a still image when mpeos_dispGetBGColor() is called, the color returned has an undefined value.

related function(s) mpeos_dispBGImageDelete
mpeos_dispBGImageGetSize
mpeos_dispBGImageNew
mpeos_dispDisplayBGImage
mpeos_dispSetBGColor

mpeos_dispGetCoherentConfigCount

gets the number of coherent screen configurations.

syntax mpe_Error mpeos_dispGetCoherentConfigCount(
 mpe_DispScreen *screen*,
 uint32_t * *nSets*);

parameter(s) *screen* specifies the screen to query. *screen* is returned from a previous call `mpeos_dispGetScreens()`. `mpe_DispScreen` is described in the `mpe_DispScreen` section.

nSets is an output pointer to the number of supported coherent configuration sets.

value returned If the call is successful, `mpeos_dispGetCoherentConfigCount()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetCoherentConfigCount()` should get the number of defined coherent configurations for the screen. This is determined by the number of supported screen modes and preferences. Standard definition and enhanced definition modes do not consider the preference combinations. However, high-definition does consider the preference combinations.

related function(s) `mpeos_dispGetCoherentConfigs`
`mpeos_dispSetCoherentConfig`

mpeos_dispGetCoherentConfigs

gets the screen configurations.

syntax mpe_Error mpeos_dispGetCoherentConfigs(
 mpe_DispScreen *screen*,
 mpe_DispCoherentConfig * *set*);

| | | |
|---------------------|---------------|--|
| parameter(s) | <i>screen</i> | specifies the screen to query. <i>screen</i> is returned from a previous call <code>mpeos_dispGetScreens()</code> . <code>mpe_DispScreen</code> is described in the <i>mpe_DispScreen</i> section. |
| | <i>set</i> | is an input pointer to the address of an array to be populated with <code>mpe_DispCoherentConfig</code> handles. It is assumed that the array is at least big enough to hold references to all coherent configuration sets. That is, at least <code>sizeof(mpe_DispCoherentConfig) * nSets</code> bytes where <i>nSets</i> can be retrieved using <code>mpeos_dispGetCoherentConfigCount()</code> . <code>mpe_DispCoherentConfig</code> is described in the <i>mpe_DispCoherentConfig</i> section. |

value returned If the call is successful, `mpeos_dispGetCoherentConfigs()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`

indicates an invalid parameter was passed to the function.

description `mpeos_dispGetCoherentConfigs()` should get the set of coherent configurations supported by the screen.

related function(s) `mpeos_dispGetCoherentConfigCount`
`mpeos_dispSetCoherentConfig`

mpeos_dispGetConfigCount

gets the number of device configurations.

syntax `mpe_Error mpeos_dispGetConfigCount(
 mpe_DispDevice device,
 uint32_t * nConfigs);`

parameter(s) `device` specifies the device to query. `device` is returned by a previous call to `mpeos_dispGetDevices()`. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

`nConfigs` is an output pointer to where the number of supported configurations is stored.

value returned If the call is successful, `mpeos_dispGetConfigCount()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetConfigCount()` should get the number of supported configurations for the device.

related function(s) `mpeos_dispGetConfigInfo`
`mpeos_dispGetConfigs`
`mpeos_dispGetConfigSet`
`mpeos_dispGetConfigSetCount`
`mpeos_dispGetCurrConfig`
`mpeos_dispSetCurrConfig`

mpeos_dispGetConfigInfo

gets configuration information.

syntax mpe_Error mpeos_dispGetConfigInfo(
 mpe_DispDeviceConfig *config*,
 mpe_DispDeviceConfigInfo * *info*);

| | | |
|---------------------|---------------|---|
| parameter(s) | <i>config</i> | specifies the configuration to query. <i>config</i> is returned by a previous call to either <code>mpeos_dispGetConfigs()</code> , <code>mpeos_dispGetConfigSet()</code> , or <code>mpeos_dispGetCurrConfig()</code> . <code>mpe_DispDeviceConfig</code> is described in the <i>mpe_DispDeviceConfig</i> section. |
| | <i>info</i> | is an output pointer to a buffer to populate with information about the configuration. <code>mpe_DispDeviceConfigInfo</code> is described in the <i>mpe_DispDeviceConfigInfo</i> section. |

value returned If the call is successful, `mpeos_dispGetConfigInfo()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetConfigInfo()` should get information about the configuration. Configuration information includes the device to which this configuration belongs. If flicker filter is supported, `mpeos_dispGetConfigInfo()` determines if the configuration is interlaced, the pixel resolution, the pixel aspect ratio, the normalized screen area covered, determines if MPEG I-frame stills are supported, and indicates if a changeable single background color is supported.

related function(s) `mpeos_dispGetConfigCount`
`mpeos_dispGetConfigs`
`mpeos_dispGetConfigSet`
`mpeos_dispGetConfigSetCount`
`mpeos_dispGetCurrConfig`
`mpeos_dispSetCurrConfig`

mpeos_dispGetConfigs

gets the device configurations.

syntax `mpe_Error mpeos_dispGetConfigs(mpe_DispDevice device, mpe_DispDeviceConfig * configs);`

parameter(s) `device` specifies the device to query. `device` is returned by a previous call to `mpeos_dispGetDevices()`. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

`configs` is an output pointer to the address of an array to be populated with `mpe_DispDeviceConfig` handles. It is assumed the array is at least big enough to hold references to all the configurations. Meaning, at least the `sizeof(mpe_DispDeviceConfig) * nConfigs` bytes, where `nConfigs` can be retrieved using `mpeos_dispGetConfigCount()`. `mpe_DispDeviceConfig` is described in the *mpe_DispDeviceConfig* section.

value returned If the call is successful, `mpeos_dispGetConfigs()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetConfigs()` should get a set of supported display device configurations for the device.

related function(s) `mpeos_dispGetConfigCount`
`mpeos_dispGetConfigInfo`
`mpeos_dispGetConfigSet`
`mpeos_dispGetConfigSetCount`
`mpeos_dispGetCurrConfig`
`mpeos_dispSetCurrConfig`

mpeos_dispGetConfigSet

gets a set of configurations.

syntax `mpe_Error mpeos_dispGetConfigSet(mpe_DispCoherentConfig set, mpe_DispDeviceConfig * configs);`

parameter(s) *set* specifies the coherent configuration set to query. *set* is returned by a previous call to `mpeos_dispGetCoherentConfigs()`.

`mpe_DispCoherentConfig` is described in the *mpe_DispCoherentConfig* section.

configs is an output pointer to the address of an array to be populated with `mpe_DispCoherentConfig` handles. It is assumed that the array is at least big enough to hold references to all configurations. That is, at least `sizeof(mpe_DispDeviceConfig) * nConfigs` bytes where *nConfigs* can be retrieved using `mpeos_dispGetConfigSetCount().mpe_DispDeviceConfig` is described in the *mpe_DispDeviceConfig* section.

value returned If the call is successful, `mpeos_dispGetConfigSet()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`

indicates an invalid parameter was passed to the function.

description `mpeos_dispGetConfigSet()` should get the set of configurations represented by the coherent configuration set. If the call is successful, it gets a graphic configuration and a video configuration that make up the coherent set.

related function(s) `mpeos_dispGetConfigCount`
`mpeos_dispGetConfigInfo`
`mpeos_dispGetConfigs`
`mpeos_dispGetConfigSetCount`
`mpeos_dispGetCurrConfig`
`mpeos_dispSetCurrConfig`

mpeos_dispGetConfigSetCount

gets the number of device configurations.

syntax `mpe_Error mpeos_dispGetConfigSetCount(`
`mpe_DispCoherentConfig set,`
`uint32_t * nConfigs);`

parameter(s) `set` specifies the coherent configuration set to query. `set` is returned by a previous call to `mpeos_dispGetCoherentConfigs()`. `mpe_DispCoherentConfig` is described in the `mpe_DispCoherentConfig` section.

`nConfigs` is an output pointer to the address where the number of configurations is written.

value returned If the call is successful, `mpeos_dispGetConfigSetCount()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
 indicates an invalid parameter was passed to the function.

description `mpeos_dispGetConfigSetCount()` should get the number of distinct device configurations represented by the coherent configuration set. If the call is successful, `mpeos_dispGetConfigSetCount()` returns the set of configurations that make up the single coherent configuration.

related function(s) `mpeos_dispGetConfigCount`
`mpeos_dispGetConfigInfo`
`mpeos_dispGetConfigs`
`mpeos_dispGetConfigSet`
`mpeos_dispGetCurrConfig`
`mpeos_dispSetCurrConfig`

mpeos_dispGetCurrConfig

gets the current device configuration.

syntax mpe_Error mpeos_dispGetCurrConfig(
 mpe_DispDevice *device*,
 mpe_DispDeviceConfig * *config*);

parameter(s) *device* specifies the device to query. *device* is returned by a previous call to `mpeos_dispGetDevices()`. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

config is an output pointer to the address where the current configuration is written. `mpe_DispDeviceConfig` is described in the *mpe_DispDeviceConfig* section.

value returned If the call is successful, `mpeos_dispGetCurrConfig()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetCurrConfig()` should get the current set of configurations for the device.

related function(s) `mpeos_dispGetConfigCount`
`mpeos_dispGetConfigInfo`
`mpeos_dispGetConfigs`
`mpeos_dispGetConfigSet`
`mpeos_dispGetConfigSetCount`
`mpeos_dispSetCurrConfig`

mpeos_dispGetDeviceCount

gets the number of supported devices.

syntax `mpe_Error mpeos_dispGetDeviceCount(
 mpe_DispScreen screen,
 mpe_DispDeviceType type,
 uint32_t * nDevices);`

| | | |
|----------------------------|---|--|
| parameter(s) | <i>screen</i> | specifies the screen to query. <i>screen</i> is returned from a previous call <code>mpeos_dispGetScreens()</code> . <code>mpe_DispScreen</code> is described in the <i>mpe_DispScreen</i> section. |
| | <i>type</i> | specifies the type of device to count. <code>mpe_DispDeviceType</code> is described in the <i>mpe_DispDeviceType</i> section. Currently, the following values are supported for <i>type</i> : |
| | <code>MPE_DISPLAY_ALL_DEVICES</code> | gets the total count of all device types for the screen. |
| | <code>MPE_DISPLAY_GRAPHICS_DEVICE</code> | indicates a graphic device type. |
| | <code>MPE_DISPLAY_VIDEO_DEVICE</code> | indicates a video device. |
| | <code>MPE_DISPLAY_BACKGROUND_DEVICE</code> | indicates a background device. |
| | <i>nDevices</i> | is an output pointer to the number of devices for the screen. |
| value returned | If the call is successful, <code>mpeos_dispGetDeviceCount()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return the following error code: | |
| | <code>MPE_DISP_ERROR_INVALID_PARAM</code> indicates an invalid parameter was passed to the function. | |
| description | <code>mpeos_dispGetDeviceCount()</code> should get the number of supported display devices (graphics, video, or background) for the display screen. | |
| related function(s) | <code>mpeos_dispGetDeviceInfo</code> <code>mpeos_dispGetDevices</code> | |

mpeos_dispGetDeviceInfo

gets device information.

syntax mpe_Error mpeos_dispGetDeviceInfo(
 mpe_DispDevice *device*,
 mpe_DispDeviceInfo * *info*);

parameter(s) *device* specifies the device to query. *device* is returned by a previous call to `mpeos_dispGetDevices()`. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

info is an output pointer to the structure filled with device information. `mpe_DispDeviceInfo` is described in the *mpe_DispDeviceInfo* section.

value returned If the call is successful, `mpeos_dispGetDeviceInfo()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetDeviceInfo()` should get information about the device. Device information includes the type of device, the identification string associated with the device, the parent screen, aspect ratio, and a bitmask representing the types of devices whose configurations might be affected if the configuration of this device is changed.

related function(s) `mpeos_dispGetDeviceCount`
`mpeos_dispGetDevices`

mpeos_dispGetDevices

gets the devices of a given type for the screen.

syntax mpe_Error mpeos_dispGetDevices(
 mpe_DispScreen *screen*,
 mpe_DispDeviceType *type*,
 mpe_DispDevice * *devices*);

| | | |
|----------------------------|-------------------------------|---|
| parameter(s) | <i>screen</i> | specifies the screen to query. <i>screen</i> is returned from a previous call <code>mpeos_dispGetScreens()</code> . <code>mpe_DispScreen</code> is described in the <i>mpe_DispScreen</i> section. |
| | <i>type</i> | specifies the type of device(s). <code>mpe_DispDeviceType</code> is described in the <i>mpe_DispDeviceType</i> section. Currently, the following values are supported for <i>type</i> : |
| | MPE_DISPLAY_ALL_DEVICES | gets the total count of all device types for the screen. |
| | MPE_DISPLAY_GRAPHICS_DEVICE | indicates a graphic device type. |
| | MPE_DISPLAY_VIDEO_DEVICE | indicates a video device. |
| | MPE_DISPLAY_BACKGROUND_DEVICE | indicates a background device. |
| | <i>devices</i> | is an output pointer to an array where devices should be written. It is assumed the array is at least big enough to hold references to all the display screens. Meaning, at least the <code>sizeof(mpe_DispDevice) * nDevices</code> bytes, where <code>nDevices</code> can be retrieved using <code>mpeos_dispGetDeviceCount()</code> . <code>mpe_DispDevice</code> is described in the <i>mpe_DispDevice</i> section. |
| value returned | | If the call is successful, <code>mpeos_dispGetDevices()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return the following error code: |
| | MPE_DISP_ERROR_INVALID_PARAM | indicates an invalid parameter was passed to the function. |
| description | | <code>mpeos_dispGetDevices()</code> should get the requested type of display devices for a screen. |
| related function(s) | | <code>mpeos_dispGetDeviceCount</code> <code>mpeos_dispGetDeviceInfo</code> |

mpeos_dispGetDFC

gets the current DFC for the specified decoder.

syntax `mpe_Error mpeos_dispGetDFC(`
`mpe_DispDevice decoder,`
`mpe_DispDfcAction * action,`
`mpe_Bool * isPlatformMode);`

| | | |
|---------------------|---|---|
| parameter(s) | <i>decoder</i> | specifies the decoder. <code>mpe_DispDevice</code> is described in the <i>mpe_DispDevice</i> section. |
| | <i>action</i> | is an output pointer to the DFC. <code>mpe_DispDfcAction</code> is described in the <i>mpe_DispDfcAction</i> section. Currently, the following values are supported for <i>action</i> : |
| | <code>MPE_DFC_PROCESSING_NONE</code> | indicates the decoder format conversion is inactive. |
| | <code>MPE_DFC_PROCESSING_FULL</code> | indicates the full 720x576 frame is transferred (this may be either 4:3 or 16:9; part of this may be black (for example, in pillar box cases). |
| | <code>MPE_DFC_PROCESSING_LB_16_9</code> | indicates the 720x576 input grid is transferred into a 16:9 letterbox format in a 4:3 frame. |
| | <code>MPE_DFC_PROCESSING_LB_14_9</code> | indicates the 720x576 input grid is transferred into a 14:9 letterbox format in a 4:3 frame. |
| | <code>MPE_DFC_PROCESSING_CCO</code> | indicates the 4:3 central part of an 720x576 input 16:9 frame is transferred into a 720x576 4:3 output frame. |
| | <code>MPE_DFC_PROCESSING_PAN_SCAN</code> | indicates no decoder format conversion. |
| | <code>MPE_DFC_PROCESSING_LB_2_21_1_ON_4_3</code> | indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 4:3 frame. |
| | <code>MPE_DFC_PROCESSING_LB_2_21_1_ON_16_9</code> | indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 16:9 frame. |
| | <code>MPE_DFC_PLATFORM</code> | indicates the decoder format conversion for the platform is used. |
| | <code>MPE_DFC_PROCESSING_16_9_ZOOM</code> | indicates the central 16:9 letterbox area of the 4:3 720x576 input grid is expanded to fill the 16:9 output frame. |

isPlatformMode

is an output pointer indicating whether or not the decoder is in platform mode. *isPlatformMode* is true if the decoder is in platform mode, false if not. *mpe_Bool* is described in the *mpe_Bool* section.

value returned If the call is successful, `mpeos_dispGetDFC()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE EINVAL` indicates *decoder* has an invalid value.

description `mpeos_dispGetDFC()` should get the current DFC for the specified decoder.

related function(s) `mpeos_dispCheckDFC`
`mpeos_dispSetDFC`

mpeos_dispGetGfxSurface

gets a handle for the screen surface for the specified device.

syntax mpe_Error mpeos_dispGetGfxSurface(
 mpe_DispDevice *device*,
 mpe_GfxSurface * *surface*);

parameter(s) *device* specifies the device. *mpe_DispDevice* is described in the *mpe_DispDevice* section.

surface is an output pointer for the screen surface. *mpe_GfxSurface* is described in the *mpe_GfxSurface* section.

value returned If the call is successful, *mpeos_dispGetGfxSurface()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *device* has an invalid value.

description *mpeos_dispGetGfxSurface()* should get a handle for the screen surface of the specified device. This surface should continue to be valid throughout all changes to the graphics device configuration (allowing the surface handle to be constant over the life of the current runtime).

-
- ◆ **NOTE:** Changes to the graphics device configuration (in particular, changes regarding pixel resolution) may result in behind-the-scenes changes to the returned surface. Configuration changes should be made through *mpeos_gfxSurfaceGetInfo()* in the MPEOS Graphics API.

related function(s) *mpeos_dispFlushGfxSurface*

mpeos_dispGetOutputPortCount

gets the number of video output ports.

syntax mpe_Error mpeos_dispGetOutputPortCount(uint32_t * nPorts);

parameter(s) *nPorts* is an output pointer to the video output ports.

value returned If the call is successful, mpeos_dispGetOutputPortCount() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_DISP_ERROR_INVALID_PARAM
indicates an invalid parameter was passed to the function.

description mpeos_dispGetOutputPortCount() should get the number of supported video output ports (irrespective of any screen association).

related function(s) mpeos_dispEnableOutputPort
mpeos_dispGetOutputPortInfo
mpeos_dispGetOutputPorts

mpeos_dispGetOutputPortInfo

gets output port information.

syntax mpe_Error mpeos_dispGetOutputPortInfo(
 mpe_DispOutputPort *port*,
 mpe_DispOutputPortInfo * *info*);

parameter(s) *port* identifies the output port. *port* is returned from a previous call to `mpeos_dispGetOutputPorts`. `mpe_DispOutputPort` is described in the *mpe_DispOutputPort* section.

info is an output pointer to the structure where the information is written. `mpe_DispOutputPortInfo` is described in the *mpe_DispOutputPortInfo* section.

value returned If the call is successful, `mpeos_dispGetOutputPortCount()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`

indicates an invalid parameter was passed to the function.

description `mpeos_dispGetOutputPortCount()` should get information about the output port. Information includes the type of the port, whether the output port is enabled, whether DTCP is supported, whether HDCP is supported, and the restricted vertical resolution of high-definition output.

related function(s) `mpeos_dispEnableOutputPort`
`mpeos_dispGetOutputPortCount`
`mpeos_dispGetOutputPorts`

mpeos_dispGetOutputPorts

gets the video output ports.

syntax `mpe_Error mpeos_dispGetOutputPorts(mpe_DispOutputPort * ports);`

parameter(s) `ports` is an output pointer to the array where handles are written. It is assumed the array is at least big enough to hold references to all output ports. That is, at least `sizeof(mpe_DispOutputPort) * nPorts` bytes where `nPorts` can be retrieved using `mpeos_dispGetOutputPortCount()`. `mpe_DispOutputPort` is described in the *mpe_DispOutputPort* section.

value returned If the call is successful, `mpeos_dispGetOutputPorts()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetOutputPorts()` should get the handles for all the supported video output ports.

related function(s) `mpeos_dispEnableOutputPort`
`mpeos_dispGetOutputPortInfo`
`mpeos_dispGetOutputPortInfo`

mpeos_dispGetRFBypassState

gets the state of the RF bypass.

syntax mpe_Error mpeos_dispGetRFBypassState(mpe_Bool * state);

parameter(s) *state* is an output pointer to the state for the RF bypass. If *state* is TRUE, RF bypass is enabled. If *state* is FALSE, RF bypass is disabled. *mpe_Bool* is described in the *mpe_Bool* section.

value returned If the call is successful, *mpeos_dispGetRFBypassState()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *state* has an invalid value.

description *mpeos_dispGetRFBypassState()* should get the current state of the RF bypass.

related function(s) *mpeos_dispSetRFBypassState*

mpeos_dispGetRFChannel

gets the RF output channel.

syntax mpe_Error mpeos_dispGetRFChannel(uint32_t * *channel*);

parameter(s) *channel* is an output pointer to the RF output channel.

value returned If the call is successful, mpeos_dispGetRFChannel() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE EINVAL indicates *channel* has an invalid value.

description mpeos_dispGetRFChannel() should get the channel number of the RF output.

related function(s) mpeos_dispGetRFBypassState
mpeos_dispSetRFBypassState
mpeos_dispSetRFChannel

mpeos_dispGetScreenCount

gets a count of all supported the display screens.

syntax mpe_Error mpeos_dispGetScreenCount(uint32_t * nScreens);

parameter(s) *nScreens* is an output pointer to the number of supported display screens.

value returned If the call is successful, mpeos_dispGetScreenCount() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_DISP_ERROR_INVALID_PARAM
indicates an invalid parameter was passed to the function.

description mpeos_dispGetScreenCount() should get a count of all supported display screens.

related function(s) mpeos_dispGetScreenInfo
mpeos_dispGetScreens

mpeos_dispGetScreenInfo

gets screen information.

syntax mpe_Error mpeos_dispGetScreenInfo (
 mpe_DispScreen *screen*,
 mpe_DispScreenInfo * *info*);

| | | |
|---------------------|---------------|--|
| parameter(s) | <i>screen</i> | specifies the screen to query. <i>screen</i> is returned from a previous call <code>mpeos_dispGetScreens()</code> . <code>mpe_DispScreen</code> is described in the <i>mpe_DispScreen</i> section. |
| | <i>info</i> | is an output pointer to a buffer to populate with information about the screen. <code>mpe_DispScreenInfo</code> is described in the <i>mpe_DispScreenInfo</i> section. |

value returned If the call is successful, `mpeos_dispGetScreenInfo()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispGetScreenInfo()` should get information about the screen. *info* identifies the presence of a non-contributing video configuration.

related function(s) `mpeos_dispGetScreenCount`
`mpeos_dispGetScreens`

mpeos_dispGetScreens

gets all the screen handles.

syntax mpe_Error mpeos_dispGetScreens(mpe_DispScreen * screens);

parameter(s) *screens* is an output pointer to an array to be populated with mpe_DispScreen handles. It is assumed the array is at least big enough to hold references to all display screens. Meaning, at least the sizeof(mpe_DispScreen) * *nScreens* bytes, where *nScreens* can be retrieved using mpeos_dispGetScreenCount(). mpe_DispScreen is described in the *mpe_DispScreen* section.

value returned If the call is successful, mpeos_dispGetScreens() should return MPE_SUCCESS. Otherwise, it should return one the following error codes:

MPE_DISP_ERROR_NO_ERROR
specifies the call was successful.

MPE_DISP_ERROR_UNKNOWN
indicates a generic error.

MPE_DISP_ERROR_NO_MEMORY
indicates you are out of memory.

MPE_DISP_ERROR_INVALID_PARAM
indicates an invalid parameter was passed to the function.

MPE_DISP_ERROR_UNIMPLEMENTED
indicates unimplemented support.

description mpeos_dispGetScreens() should get all the screen handles.

related function(s) mpeos_dispGetScreenCount
mpeos_dispGetScreenInfo

mpeos_dispGetVideoOutputPortOption

gets the value for a display option.

syntax `mpe_Error mpeos_dispGetVideoOutputPortOption(
 mpe_DispOutputPort port,
 mpe_DispOutputPortOption * opt);`

| | | |
|---------------------|---|---|
| parameter(s) | <i>port</i> | specifies the video output port. <i>screen</i> is returned from a previous call <code>mpeos_dispGetVideoOutputPorts()</code> . <code>mpe_DispOutputPort</code> is described in the <i>mpe_DispOutputPort</i> section. |
| | <i>opt</i> | is an output pointer to the options. <code>mpe_DispOutputPortOption</code> is described in the <i>mpe_DispOutputPortOption</i> section. Currently, the following values are defined for <i>opt</i> : |
| | <code>MPE_DISP_1394_SELECT_JACK</code> | gets the type of video input port for the display. Currently, the following values are supported: |
| | <code>MPE_DISP_1394_SELECT_JACK_TYPE_ANALOG</code> | specifies an analog jack. |
| | <code>MPE_DISP_1394_SELECT_JACK_TYPE_DIGITAL</code> | specifies a digital jack. |
| | <code>MPE_DISP_1394_DEVICE_LIST</code> | gets a list of all the devices connected to the set-top box (for example, storage device or video recorder). |
| | <code>MPE_DISP_1394_MODEL_NAME</code> | gets the model name for the current display device. |
| | <code>MPE_DISP_1394_VENDOR_NAME</code> | gets the vendor name for the current display device. |
| | <code>MPE_DISP_1394_SUBUNIT_TYPE</code> | gets the sub-unit type for the current display device. |
| | <code>MPE_DISP_1394_SELECT_SINK</code> | gets the sink node for stream connection to the current display. |

value returned If the call is successful, `mpeos_dispGetVideoOutputPortOption()` should return `MPE_SUCCESS`. Otherwise, it should return one the following error codes:

- `MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.
- `MPE_DISP_ERROR_UNIMPLEMENTED`
indicates unimplemented support.

description mpeos_dispGetVideoOutputPortOption() should get the value of an option on a video output port.

related function(s) mpeos_dispSetVideoOutputPortOption

mpeos_dispSetBGColor

sets the background color.

syntax mpe_Error mpeos_dispSetBGColor(
 mpe_DispDevice *device*,
 mpe_GfxColor *color*);

parameter(s) *device* specifies the device to display the background *color*. *device* is an MPE_DISPLAY_BACKGROUND_DEVICE type that is returned by a previous call to `mpeos_dispGetDevices()`. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

color specifies the background color. `mpe_GfxColor` is described in the *mpe_GfxColor* section.

value returned If the call is successful, `mpeos_dispSetBGColor()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispSetBGColor()` should set the background color on the specified device. The alpha channel in the color is ignored and treated as fully opaque (255). If the device is currently displaying a still image when `mpeos_dispSetBGColor()` is called, the image is cleared.

related function(s) `mpeos_dispBGImageDelete`
`mpeos_dispBGImageGetSize`
`mpeos_dispBGImageNew`
`mpeos_dispDisplayBGImage`
`mpeos_dispGetBGColor`

mpeos_dispSetCoherentConfig

sets the screen configurations.

syntax mpe_Error mpeos_dispSetCoherentConfig(
 mpe_DispScreen *screen*,
 mpe_DispCoherentConfig *set*);

| | | |
|---------------------|---------------|--|
| parameter(s) | <i>screen</i> | specifies the screen to modify. <i>screen</i> is returned from a previous call <code>mpeos_dispGetScreens()</code> . <code>mpe_DispScreen</code> is described in the <code>mpe_DispScreen</code> section. |
| | <i>set</i> | specifies the set of configurations. <i>set</i> is returned by a previous call to <code>mpeos_dispGetCoherentConfigs()</code> . <code>mpe_DispCoherentConfig</code> is described in the <code>mpe_DispCoherentConfig</code> section. |

value returned If the call is successful, `mpeos_dispSetCoherentConfig()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`

indicates an invalid parameter was passed to the function.

description `mpeos_dispSetCoherentConfig()` should apply a coherent configuration set to the specified screen. All devices of the screen with configurations contained in *set* are configured to reflect *set*.

related function(s) `mpeos_dispGetCoherentConfigCount`
`mpeos_dispGetCoherentConfigs`

mpeos_dispSetCurrConfig

sets the device configuration.

syntax `mpe_Error mpeos_dispSetCurrConfig(
 mpe_DispDevice device,
 mpe_DispDeviceConfig config);`

parameter(s) *device* specifies the device to set. *device* is returned by a previous call to `mpeos_dispGetDevices()`. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

config specifies the new configuration for the device. *config* is returned by a previous call to `mpeos_dispGetConfigs()`, `mpeos_dispGetConfigSet()`, or `mpeos_dispGetCurrConfig()`. `mpe_DispDeviceConfig` is described in the *mpe_DispDeviceConfig* section.

value returned If the call is successful, `mpeos_dispSetCurrConfig()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispSetCurrConfig()` should set the configuration for the specified device. On a successful return, the change to the device configuration should be considered complete. If the setting requires changes to other device configurations, the call fails. If the current configuration is the same as the new configuration, this operation should have no effect.

related function(s) `mpeos_dispGetConfigCount`
`mpeos_dispGetConfigInfo`
`mpeos_dispGetConfigs`
`mpeos_dispGetConfigSet`
`mpeos_dispGetConfigSetCount`
`mpeos_dispGetCurrConfig`

mpeos_dispSetDFC

sets the DFC for the decoder.

syntax mpe_Error mpeos_dispSetDFC(
 mpe_DispDevice *decoder*,
 mpe_DispDfcAction *action*);

| | | |
|---------------------|--------------------------------------|---|
| parameter(s) | <i>decoder</i> | specifies the decoder. <i>mpe_DispDevice</i> is described in the <i>mpe_DispDevice</i> section. |
| | <i>action</i> | specifies the DFC for the decoder. <i>mpe_DispDfcAction</i> is described in the <i>mpe_DispDfcAction</i> section. Currently, the following values are supported for <i>action</i> : |
| | MPE_DFC_PROCESSING_NONE | indicates the decoder format conversion is inactive. |
| | MPE_DFC_PROCESSING_FULL | indicates the full 720x576 frame is transferred (this may be either 4:3 or 16:9; part of this may be black (for example, in pillar box cases)). |
| | MPE_DFC_PROCESSING_LB_16_9 | indicates the 720x576 input grid is transferred into a 16:9 letterbox format in a 4:3 frame. |
| | MPE_DFC_PROCESSING_LB_14_9 | indicates the 720x576 input grid is transferred into a 14:9 letterbox format in a 4:3 frame. |
| | MPE_DFC_PROCESSING_CCO | indicates the 4:3 central part of an 720x576 input 16:9 frame is transferred into a 720x576 4:3 output frame. |
| | MPE_DFC_PROCESSING_PAN_SCAN | indicates no decoder format conversion. |
| | MPE_DFC_PROCESSING_LB_2_21_1_ON_4_3 | indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 4:3 frame. |
| | MPE_DFC_PROCESSING_LB_2_21_1_ON_16_9 | indicates the 720x576 input grid is transferred into a 2.21:1 letterbox in a 16:9 frame. |
| | MPE_DFC_PLATFORM | indicates the decoder format conversion for the platform is used. |
| | MPE_DFC_PROCESSING_16_9_ZOOM | indicates the central 16:9 letterbox area of the 4:3 720x576 input grid is expanded to fill the 16:9 output frame. |

value returned If the call is successful, `mpeos_dispSetDFC()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_ERROR_MEDIA_OS`

specifies an error occurred in the operating system.

description `mpeos_dispSetDFC()` should set the DFC for the decoder. The `MPE_DFC_CHANGED` event should be sent to the event queue associated with a decode session for the given decoder, if any.

related function(s) `mpeos_dispCheckDFC`
`mpeos_dispGetDFC`

mpeos_dispSetRFBypassState

sets the RF bypass state.

syntax mpe_Error mpeos_dispSetRFBypassState(mpe_Bool enable);

parameter(s) *enable* determines whether RF bypass is enabled. If *enable* is TRUE, the RF bypass is enabled. If *enable* is FALSE, RF bypass is disabled. *mpe_Bool* is described in the *mpe_Bool* section.

value returned If the call is successful, `mpeos_dispSetRFBypassState()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispSetRFBypassState()` should enable the current state of RF bypass.

related function(s) `mpeos_dispGetRFChannel`
`mpeos_dispSetRFChannel`

mpeos_dispSetRFChannel

sets the RF output channel.

syntax mpe_Error mpeos_dispSetRFChannel(uint32_t *channel*);

parameter(s) *channel* specifies the channel number.

value returned If the call is successful, mpeos_dispSetRFChannel() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE EINVAL indicates *channel* has an invalid value.

description mpeos_dispSetRFChannel() should set the RF output channel.

related function(s) mpeos_dispGetRFBypassState
mpeos_dispGetRFChannel
mpeos_dispSetRFBypassState

mpeos_dispSetVideoOutputPortOption

sets the option on a video output port

syntax mpe_Error mpeos_dispSetVideoOutputPortOption(
 mpe_DispOutputPort *port*,
 mpe_DispOutputPortOption * *opt*);

| | | |
|---------------------|--|--|
| parameter(s) | <i>port</i> | specifies the video output port on which to set an option. If NULL, the set operation is done for all output ports of the same type. <i>screen</i> is returned from a previous call <code>mpeos_dispGetOutputPorts()</code> . <code>mpe_DispOutputPort</code> is described in the <i>mpe_DispOutputPort</i> section. |
| | <i>opt</i> | is an input pointer to the option. <code>mpe_DispOutputPortOption</code> is described in the <i>mpe_DispOutputPortOption</i> section. Currently, the following values are defined for <i>opt</i> : |
| | MPE_DISP_1394_SELECT_JACK | sets the type of video input port for the display. Currently, the following values are supported: |
| | MPE_DISP_1394_SELECT_JACK_TYPE_ANALOG | specifies an analog jack. |
| | MPE_DISP_1394_SELECT_JACK_TYPE_DIGITAL | specifies a digital jack. |
| | MPE_DISP_1394_DEVICE_LIST | sets a list of all the devices connected to the set-top box (for example, storage device or video recorder). |
| | MPE_DISP_1394_MODEL_NAME | sets the model name for the current display device. |
| | MPE_DISP_1394_VENDOR_NAME | sets the vendor name for the current display device. |
| | MPE_DISP_1394_SUBUNIT_TYPE | sets the sub-unit type for the current display device. |
| | MPE_DISP_1394_SELECT_SINK | sets the sink node for stream connection to the current display. |

value returned If the call is successful, `mpeos_dispSetVideoOutputPortOption()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

- MPE_DISP_ERROR_INVALID_PARAM
indicates an invalid parameter was passed to the function.
- MPE_DISP_ERROR_UNIMPLEMENTED
indicates the requested operation is not implemented.

description mpeos_dispSetVideoOutputPortOption() should set an option on a video output port.

related function(s) mpeos_dispGetVideoOutputPortOption

mpeos_dispWouldImpact

identifies the impact on other devices.

```
syntax mpe_Error mpeos_dispWouldImpact(
    mpe_DispDevice device,
    mpe_DispDeviceConfig config,
    mpe_DispDevice device2,
    mpe_DispDeviceConfig config2,
    mpe_Bool * impact );
```

| | | |
|---------------------|----------------|---|
| parameter(s) | <i>device</i> | specifies the device to query. <i>device</i> is returned by a previous call to <code>mpeos_dispGetDevices()</code> . <code>mpe_DispDevice</code> is described in the <i>mpe_DispDevice</i> section. |
| | <i>config</i> | specifies the configuration to test. <i>config</i> is returned by a previous call to <code>mpeos_dispGetConfigs()</code> , <code>mpeos_dispGetConfigSet()</code> , or <code>mpeos_dispGetCurrConfig()</code> . <code>mpe_DispDeviceConfig</code> is described in the <i>mpe_DispDeviceConfig</i> section. |
| | <i>device2</i> | specifies a second device to query. <i>device</i> is returned by a previous call to <code>mpeos_dispGetDevices()</code> . <code>mpe_DispDevice</code> is described in the <i>mpe_DispDevice</i> section. |
| | <i>config2</i> | specifies a second configuration to test. <i>config2</i> is returned by a previous call to <code>mpeos_dispGetConfigs()</code> , <code>mpeos_dispGetConfigSet()</code> , or <code>mpeos_dispGetCurrConfig()</code> . <code>mpe_DispDeviceConfig</code> is described in the <i>mpe_DispDeviceConfig</i> section. |
| | <i>impact</i> | is an output pointer to the comparison results. If <i>impact</i> is TRUE, the configurations are coherent. If <i>impact</i> is FALSE, the configurations are not coherent. <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |

value returned If the call is successful, `mpeos_dispWouldImpact()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_DISP_ERROR_INVALID_PARAM`
indicates an invalid parameter was passed to the function.

description `mpeos_dispWouldImpact()` should compare the configurations for both devices and determine if the configurations are coherent or if they impact other devices.

related function(s) `mpeos_dispSetBGColor`
`mpeos_dispSetCoherentConfig`
`mpeos_dispSetCurrConfig`
`mpeos_dispSetRFBypassState`
`mpeos_dispSetRFChannel`

DVR Digital Video Recorder (DVR) API

Overview

The DVR Application Programming Interface (API) provides functions to create and playback digital video recordings. It also provides time-shift functionality that enables record, pause, rewind, and fast-forward of real-time broadcast content.

The DVR API provides support to the OpenCable Application Platform (OCAP) DVR package implementation, as well as DVR specific Java Media Framework (JMF) Player controls. These APIs can be directly mapped to the Multimedia Platform Environment Operating System (MPEOS) APIs porting layer which can be divided into the following categories:

- ◆ playback and trick modes (controls speed and direction)
- ◆ time-shift buffer (live and scheduled recording)
- ◆ record information (for querying the list of recordings, available disk space, etc.)

Before reading this chapter, you should be familiar with:

- ◆ what the DVR functions are and how they work
- ◆ the DVR requirements described in the OpenCable Application Platform Specification OCAP Digital Video Recorder (DVR)

After reading this chapter, you should be able to port the DVR functionality within the OCAP stack

Definitions

The following definitions, which are defined in `mpeos_dvr.h`, are used by the DVR functions:

| | |
|--|----------------------------------|
| <code>MPE_DVR_MAX_NAME_SIZE</code> | <code>MPE_DVR_MAX_PIDS</code> |
| <code>MPE_DVR_MEDIA_VOL_MAX_PATH_SIZE</code> | <code>MPE_DVR_PID_UNKNOWN</code> |
| <code>MPE_DVR_POSITIVE_INFINITY</code> | |

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPE_DVR_MAX_NAME_SIZE`

`MPE_MAX_NAME_SIZE` specifies the maximum length of a recording name. `MPE_MAX_NAME_SIZE` is described as follows:

```
#define MPE_DVR_MAX_NAME_SIZE OS_DVR_MAX_NAME_SIZE
```

`MPE_DVR_MAX_PIDS`

`MPE_DVR_MAX_PIDS` specifies the maximum number of PIDs allowed in the `mpe_DvrPidTable`. `MPE_DVR_MAX_PIDS` is described as follows:

```
#define MPE_DVR_MAX_PIDS (10)
```

`MPE_DVR_MEDIA_VOL_MAX_PATH_SIZE`

`MPE_DVR_MEDIA_VOL_MAX_PATH_SIZE` specifies the maximum number of characters in the path for a Media Storage Volume (MSV).

`MPE_DVR_MEDIA_VOL_MAX_PATH_SIZE` is described as follows:

```
#define MPE_DVR_MEDIA_VOL_MAX_PATH_SIZE  
OS_DVR_MEDIA_VOL_MAX_PATH_SIZE
```

`MPE_DVR_PID_UNKNOWN`

`MPE_DVR_PID_UNKNOWN` specifies the PID is unknown. `MPE_DVR_PID_UNKNOWN` is described as follows:

```
#define MPE_DVR_PID_UNKNOWN (-1)
```

`MPE_DVR_POSITIVE_INFINITY`

`MPE_DVR_POSITIVE_INFINITY` specifies the maximum length of a recording name. `MPE_DVR_POSITIVE_INFINITY` is described as follows:

```
#define MPE_DVR_POSITIVE_INFINITY OS_DVR_POSITIVE_INFINITY
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h`, `mpeos_dvr.h`, `mpeos_event.h`, `mpeos_media.h`, `mpeos_si.h`, and `mpeos_storage.h` are used by the DVR functions:

| | |
|-----------------------------------|---|
| <code>mpe_Bool</code> | <code>mpe_DispDevice</code> |
| <code>mpe_DvrBitRate</code> | <code>mpe_DvrBuffering</code> |
| <code>mpe_DvrConversion</code> | <code>mpe_DvrError</code> |
| <code>mpe_DvrEvent</code> | <code>mpe_DvrInfoParam</code> |
| <code>mpe_DvrPidInfo</code> | <code>mpe_DvrPidTable</code> |
| <code>mpe_DvrPlayback</code> | <code>mpe_DvrState</code> |
| <code>mpe_DvrString_t</code> | <code>mpe_DvrTsb</code> |
| <code>mpe_Error</code> | <code>mpe_EventQueue</code> |
| <code>mpe_MediaVolume</code> | <code>mpe_MediaVolumeAllowedList</code> |
| <code>mpe_MediaVolumeEvent</code> | <code>mpe_MediaVolumeInfoParam</code> |
| <code>mpe_SiElemStreamType</code> | <code>mpe_Storage</code> |
| <code>mpe_StorageHandle</code> | <code>mpeos_MediaVolume</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is described in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_DispDevice`

`mpe_DispDevice` is a pointer to a device handle. It serves as an abstraction for a screen device belonging to a single screen. The screen is described as video, graphics, or background type. `mpe_DispDevice` is described in `mpeos_display.h` as follows:

```
typedef struct { int unused1; } * mpe_DispDevice;
```

`mpe_DvrBitRate`

`mpe_DvrBitRate` defines the recording quality for the DVR functions. `mpe_DvrBitRate` is described in `mpeos_dvr.h` as follows:

```
typedef enum mpe_DvrBitRate {
    MPE_DVR_BITRATE_LOW = 6000,
    MPE_DVR_BITRATE_MEDIUM = 12000,
    MPE_DVR_BITRATE_HIGH = 19500
} mpe_DvrBitRate;
```

where

`MPE_DVR_BITRATE_LOW`
enables the lowest possible bit rate.

`MPE_DVR_BITRATE_MEDIUM`
enables a moderate bit rate.

`MPE_DVR_BITRATE_HIGH`
enables the highest bit rate for high-definition boxes.

`mpe_DvrBuffering`

`mpe_DvrBuffering` is an opaque handle for the buffering session data structure. `mpe_DvrBuffering` is described in `mpeos_dvr.h` as follows:

```
typedef struct _mpe_DvrBufferingH { int unused1; }
*mpe_DvrBuffering;
```

mpe_DvrConversion

`mpe_DvrConversion` is a pointer to the DVR handle for a convert-recording session. `mpe_DvrConversion` is described in `mpeos_dvr.h` as follows:

```
typedef struct _mpe_DvrConversionH { int unused1; }
*mpe_DvrConversion;
```

mpe_DvrError

`mpe_DvrError` defines the error codes returned by the DVR functions. `mpe_DvrError` is described in `mpeos_dvr.h` as follows:

```
typedef enum mpe_DvrError {
    MPE_DVR_ERR_NOERR,
    MPE_DVR_ERR_INVALID_PID,
    MPE_DVR_ERR_INVALID_PARAM,
    MPE_DVR_ERR_OS_FAILURE,
    MPE_DVR_ERR_PATH_ENGINE,
    MPE_DVR_ERR_UNSUPPORTED,
    MPE_DVR_ERR_NOT_ALLOWED,
    MPE_DVR_ERR_DEVICE_ERR,
    MPE_DVR_ERR_OUT_OF_SPACE,
    MPE_DVR_ERR_NOT_IMPLEMENTED,
    MPE_DVR_NO_ACTIVE_SESSION,
} mpe_DvrError;
```

where

`MPE_DVR_ERR_NOERR`

indicates no DVR error occurred.

`MPE_DVR_ERR_INVALID_PID`

indicates an invalid Packet Identifier (PID) error occurred.

`MPE_DVR_ERR_INVALID_PARAM`

indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`

indicates an error occurred at the operating-system level.

`MPE_DVR_ERR_PATH_ENGINE`

indicates an error occurred while calling the path engine.

`MPE_DVR_ERR_UNSUPPORTED`

indicates the operation is not supported.

`MPE_DVR_ERR_NOT_ALLOWED`

indicates the operation is not allowed.

`MPE_DVR_ERR_DEVICE_ERR`

indicates a hardware device error.

`MPE_DVR_ERR_OUT_OF_SPACE`

indicates no more space on the disk drive.

`MPE_DVR_ERR_NOT_IMPLEMENTED`

indicates DVR functionality is not implemented.

`MPE_DVR_NO_ACTIVE_SESSION`

indicates there is no active DVR session for this operation.

mpe_DvrInfoParam

`mpe_DvrInfoParam` defines the recording parameters for the DVR functions. `mpe_DvrInfoParam` is described in `mpeos_dvr.h` as follows:

```
typedef enum mpe_DvrInfoParam {
    MPE_DVR_MEDIA_TIME,
    MPE_DVR_TSB_START_TIME,
    MPE_DVR_TSB_END_TIME,
    MPE_DVR_TSB_MIN_BUF_SIZE,
    MPE_DVR_RECORDING_SIZE,
    MPE_DVR_RECORDING_LENGTH,
    MPE_DVR_RECORDING_MSV,
    MPE_DVR_RECORDING_NAME,
    MPE_DVR_RECORDING_QUALITY,
    MPE_DVR_RECORDING_COUNT,
    MPE_DVR_RECORDING_LIST,
    MPE_DVR_PID_INFO,
    MPE_DVR_PID_COUNT,
    MPE_DVR_STORAGE_SPACE,
    MPE_DVR_STATUS,
    MPE_DVR_MAX_BITRATE,
    MPE_DVR_MAX_RECORDING_BANDWIDTH,
    MPE_DVR_MAX_PLAYBACK_BANDWIDTH,
    MPE_DVR_STORAGE_TOTAL_CAPACITY,
    MPE_DVR_STORAGE_AVFS_CAPACITY,
    MPE_DVR_SIMULTANEOUS_PLAY_RECORD,
    MPE_DVR_STORAGE_MEDIAFS_CAPACITY,
    MPE_DVR_STORAGE_MEDIAFS_ALLOCATABLE_SPACE,
    MPE_DVR_STORAGE_MEDIAFS_FREE_SPACE,
    MPE_DVR_SUPPORTS_CROSS_MSV_TSB_CONVERT,
    MPE_DVR_MEDIA_START_TIME,
} mpe_DvrInfoParam;
```

where

MPE_DVR_MEDIA_TIME

indicates the current position in the media stream in nanoseconds from the beginning of the recording and is used by `mpeos_dvrRecordingGet()`.

MPE_DVR_TSB_START_TIME

indicates the beginning of the buffer in nanoseconds. If the buffer has not wrapped around or a position within a valid area of the buffer after the wrap around, `MPE_DVR_TSB_START_TIME` is equal to 0.

MPE_DVR_TSB_END_TIME

indicates the end of the buffer in nanoseconds. The end time represents how long the time-shift buffer has been in the `MPE_DVR_RECORDING` state.

MPE_DVR_TSB_MIN_BUF_SIZE

indicates the minimum time-shift buffer size.

MPE_DVR_RECORDING_SIZE

indicates the size of both the buffer and recording in bytes.

- MPE_DVR_RECORDING_LENGTH**
indicates the recording length in seconds.
- MPE_DVR_RECORDING_MSV**
indicates the media storage volume where the recording is stored.
- MPE_DVR_RECORDING_NAME**
indicates the unique-name identifier for the recording.
- MPE_DVR_RECORDING_QUALITY**
indicates the quality of a recording.
- MPE_DVR_RECORDING_COUNT**
indicates the number of stored recordings.
- MPE_DVR_RECORDING_LIST**
displays a list of stored recordings.
- MPE_DVR_PID_INFO**
indicates the PID information.
- MPE_DVR_PID_COUNT**
indicates the number of PIDs.
- MPE_DVR_STORAGE_SPACE**
indicates the space left on a storage device.
- MPE_DVR_STATUS**
indicates the DVR system status.
- MPE_DVR_MAX_BITRATE**
indicates the maximum supported bit rate for recording.
- MPE_DVR_MAX_RECORDING_BANDWIDTH**
indicates the maximum recording bandwidth.
- MPE_DVR_MAX_PLAYBACK_BANDWIDTH**
indicates the maximum playback bandwidth.
- MPE_DVR_STORAGE_TOTAL_CAPACITY**
indicates the total capacity of the storage device.
- MPE_DVR_STORAGE_AVFS_CAPACITY**
indicates the storage capacity for Audio Video File System (AVFS) content.
- MPE_DVR_SIMULTANEOUS_PLAY_RECORD**
indicates simultaneous play and record.
- MPE_DVR_STORAGE_MEDIAFS_CAPACITY**
indicates the storage capacity for media, file-system (MEDIAFS) content across all media storage volume.
- MPE_DVR_STORAGE_MEDIAFS_ALLOCATABLE_SPACE**
indicates the free MEDIAFS on the storage device not yet reserved by any media storage volume.
- MPE_DVR_STORAGE_MEDIAFS_FREE_SPACE**
indicates the free MEDIAFS on the storage device including any unused space reserved by media storage volume.

MPE_DVR_SUPPORTS_CROSS_MSV_TSB_CONVERT

indicates whether the platform supports the ability to convert TSB content located in one media storage volume to a linear recording in different MSV.

MPE_DVR_MEDIA_START_TIME

indicates the recording media start time.

mpe_DvrMediaStreamType

`mpe_DvrMediaStreamType` specifies the type of media stream.

`mpe_DvrMediaStreamType` is defined as follows:

```
typedef enum mpe_DvrMediaStreamType {
    MPE_DVR_MEDIA_UNKNOWN,
    MPE_DVR_MEDIA_VIDEO,
    MPE_DVR_MEDIA_AUDIO,
    MPE_DVR_MEDIA_DATA,
    MPE_DVR_MEDIA_SUBTITLES,
    MPE_DVR_MEDIA_SECTIONS,
    MPE_DVR_MEDIA_PCR
} mpe_DvrMediaStreamType;
```

where

MPE_DVR_MEDIA_UNKNOWN

specifies an unknown media type.

MPE_DVR_MEDIA_VIDEO

specifies a video stream.

MPE_DVR_MEDIA_AUDIO

specifies an audio stream.

MPE_DVR_MEDIA_DATA

specifies a data stream.

MPE_DVR_MEDIA_SUBTITLES

specifies a subtitle stream.

MPE_DVR_MEDIA_SECTIONS

specifies the sections.

MPE_DVR_MEDIA_PCR

specifies a Program Clock Reference (PCR) stream.

mpe_DvrPidInfo

`mpe_DvrPidInfo` specifies the DVR PID information. `mpe_DvrPidInfo` is defined as follows:

```
typedef struct _mpe_DvrPidInfo {
    mpe_DvrMediaStreamType streamType;
    int16 srcPid;
    int16 recPid;
    mpe_SIElemStreamType srcEltStreamType;
    mpe_SIElemStreamType recEltStreamType;
} mpe_DvrPidInfo;
```

where

streamType specifies the audio, video, data, Program Clock Reference (PCR), etc.

srcPid specifies the requested PID.

recPid specifies the actual recorded PID.

srcEltStreamType

specifies screen elements, for example, MPEG1, MPEG2, MHEG, etc. as defined in `mpeos_si.h`.

recEltStreamType

specifies recording elements, for example, MPEG1, MPEG2, MHEG, etc. as defined in `mpeos_si.h`.

mpe_DvrPidTable

`mpe_DvrPidTable` specifies the DVR PID table containing the DVR PID set and PID information. `mpe_DvrPidTable` is defined as follows:

```
typedef struct _mpe_DvrPidTable {
    int64 mediaTime;
    uint32_t count;
    mpe_DvrPidInfo pids[MPE_DVR_MAX_PIDS];
} mpe_DvrPidTable;
```

where

mediaTime specifies the media time when the PIDs are set.

count specifies the number of PIDs.

pids specifies the array with the actual recorded PIDs.

mpe_DvrPlayback

`mpe_DvrPlayback` is an opaque handle for the native playback data structure. The handle is used to control the rate and position of the playback. `mpe_DvrPlayback` is described in `mpeos_dvr.h` as follows:

```
typedef struct _mpe_DvrPlaybackH { int unused1; }
* mpe_DvrPlayback;
```

mpe_DvrState

`mpe_DvrState` defines the current state of the DVR functions. `mpe_DvrState` is described in `mpeos_dvr.h` as follows:

```
typedef enum mpe_DvrState {
    MPE_DVR_RECORDING,
    MPE_DVR_PLAYING,
    MPE_DVR_STOPPED,
    MPE_DVR_PAUSED,
} mpe_DvrState;
```

where

MPE_DVR_RECORDING

indicates the DVR is currently recording.

MPE_DVR_PLAYING

indicates the DVR is currently playing.

MPE_DVR_STOPPED

indicates the DVR buffering or playback session has stopped.

MPE_DVR_PAUSED

indicates the DVR playback is paused.

mpe_DvrString_t

`mpe_DvrString_t` specifies the name of the recording. `mpe_DvrString_t` is described in `mpeos_dvr.h` as follows:

```
typedef char mpe_DvrString_t[MAX_NAME_SIZE];
```

mpe_DvrTsb

`mpe_DvrTsb` is a pointer to the handle for the time-shift buffer. `mpe_DvrTsb` is described in `mpeos_dvr.h` as follows:

```
typedef struct _mpe_dvrTsbH { int unused1; } * mpe_DvrTsb;
```

mpe_Error

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is described in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

mpe_EventQueue

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is described in the `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

mpe_MediaVolume

`mpe_MediaVolume` provides a platform-independent handle or reference to a particular media storage volume. `mpe_MediaVolume` is described in the `mpeos_dvr.h` as follows:

```
typedef struct mpeos_MediaVolume* mpe_MediaVolume;
```

mpe_MediaVolumeAllowedList

`mpe_MediaVolumeAllowedList` provides a the list of strings identifying organizations allowed read/write access to a particular Media Storage Volume (MSV). `mpe_MediaVolumeAllowedList` is described in the `mpeos_dvr.h` as follows:

```
typedef struct {
    uint32_t numOrganizations;
    uint32_t* organizations;
} mpe_MediaVolumeAllowedList;
```

where

numOrganizations

specifies the number of organizations. On input, this field indicates the number of pre-allocated entries in the `organizations` field.

`organizations` specifies the array of organization identifiers. The memory for this array is preallocated by the caller.

mpe_MediaVolumeEvent

`mpe_MediaVolumeEvent` specifies the set of possible media volume events that may be delivered to the queue registered via `mpeos_dvrMediaVolumeRegisterQueue()`. `mpe_MediaVolumeEvent` is described in the `mpeos_dvr.h` as follows:

```
typedef enum {
    MPE_MEDIA_VOL_EVT_FREE_SPACE_ALARM = 0x1100
} mpe_MediaVolumeEvent;
```

where

`MPE_MEDIA_VOL_EVT_FREE_SPACE_ALARM`
specifies the media storage volume is low on free space.

mpe_MediaVolumeInfoParam

`mpe_MediaVolumeInfoParam` provides the set of possible media volume attributes that may be get or set via the `mpeos_dvrMediaVolumeGetInfo()` and `mpeos_dvrMediaVolumeSetInfo()` respectively.

`mpe_MediaVolumeInfoParam` is described in the `mpeos_dvr.h` as follows:

```
typedef enum mpe_MediaVolumeInfoParam {
    MPE_DVR_MEDIA_VOL_SIZE,
    MPE_DVR_MEDIA_VOL_FREE_SPACE,
    MPE_DVR_MEDIA_VOL_PATH,
    MPE_DVR_MEDIA_VOL_ALLOWED_LIST_COUNT,
    MPE_DVR_MEDIA_VOL_ALLOWED_LIST
} mpe_MediaVolumeInfoParam;
```

where

`MPE_DVR_MEDIA_VOL_SIZE`
specifies the size of the media storage volume.

`MPE_DVR_MEDIA_VOL_FREE_SPACE`
gets the size of the free space remaining in the media storage volume.

`MPE_DVR_MEDIA_VOL_PATH`
gets the path to the media storage volume.

`MPE_DVR_MEDIA_VOL_ALLOWED_LIST_COUNT`
gets the number of media storage volume in the list.

`MPE_DVR_MEDIA_VOL_ALLOWED_LIST`
specifies the number of media storage volume allowed on the list.

mpe_SiElemStreamType

`mpe_SiElemStreamType` defines values for the elementary stream_type field in the Program Map Table (PMT). On some platforms, elementary steam type definitions can be mapped to PID types directly. Otherwise, they need to be manually mapped to what the operating system requires.
`mpe_SiElemStreamType` is described in `mpeos_si.h` as follows:

```
typedef enum mpe_SiElemStreamType {
    MPE_SI_ELEM_MPEG_1_VIDEO = 0x01,
    MPE_SI_ELEM_MPEG_2_VIDEO = 0x02,
    MPE_SI_ELEM_MPEG_1_AUDIO = 0x03,
    MPE_SI_ELEM_MPEG_2_AUDIO = 0x04,
    MPE_SI_ELEM_MPEG_PRIVATE_SECTION = 0x05,
    MPE_SI_ELEM_MPEG_PRIVATE_DATA = 0x06,
    MPE_SI_ELEM_MHEG = 0x07,
    MPE_SI_ELEM_DSM_CC = 0x08,
    MPE_SI_ELEM_H_222 = 0x09,
    MPE_SI_ELEM_DSM_CC_MPE = 0x0A,
    MPE_SI_ELEM_DSM_CC_UN = 0x0B,
    MPE_SI_ELEM_DSM_CC_STREAM_DESCRIPTORS = 0x0C,
    MPE_SI_ELEM_DSM_CC_SECTIONS = 0x0D,
    MPE_SI_ELEM_AUXILIARY = 0x0E,
    MPE_SI_ELEM_VIDEO_DCII = 0x80,
    MPE_SI_ELEM_ATSC_AUDIO = 0x81,
    MPE_SI_ELEM_STD_SUBTITLE = 0x82,
    MPE_SI_ELEM_ISOCHRONOUS_DATA = 0x83,
    MPE_SI_ELEM_ASYNCHRONOUS_DATA = 0x84
} mpe_SiElemStreamType;
```

where

- MPE_SI_ELEM_MPEG_1_VIDEO
specifies Moving Picture Experts Group - video Compact Disk (CD) (MPEG-1) video.
- MPE_SI_ELEM_MPEG_2_VIDEO
specifies Moving Picture Experts Group - broadcast signals (MPEG-2) video.
- MPE_SI_ELEM_MPEG_1_AUDIO
specifies MPEG-1 audio.
- MPE_SI_ELEM_MPEG_2_AUDIO
specifies MPEG-2 audio.
- MPE_SI_ELEM_MPEG_PRIVATE_SECTION
specifies MPEG private section.
- MPE_SI_ELEM_MPEG_PRIVATE_DATA
specifies MPEG private data.
- MPE_SI_ELEM_MHEG
specifies Multimedia and Hypermedia information coding Expert Group (MHEG).

MPE_SI_ELEM_DSM_CC
 specifies Digital Storage Media - Command and Control (DSM-CC).

MPE_SI_ELEM_H_222
 specifies the international standard, Recommendation H.222.0.

MPE_SI_ELEM_DSM_CC_MPE
 specifies DSM-CC MPE.

MPE_SI_ELEM_DSM_CC_UN
 specifies Digital Storage Media - Command and Control User-to-Network (DSM-CC UN).

MPE_SI_ELEM_DSM_CC_STREAM_DESCRIPTOR
 specifies DSM-CC stream descriptors.

MPE_SI_ELEM_DSM_CC_SECTIONS
 specifies DSM-CC sections.

MPE_SI_ELEM_AUXILIARY
 specifies auxiliary.

MPE_SI_ELEM_VIDEO_DCII
 specifies video Digicipher II (DCII).

MPE_SI_ELEM_ATSC_AUDIO
 specifies Advanced Television Systems Committee (ATSC) audio.

MPE_SI_ELEM_STD_SUBTITLE
 specifies standard subtitle.

MPE_SI_ELEM_ISOCRONOUS_DATA
 specifies synchronous data.

MPE_SI_ELEM_ASYNCHRONOUS_DATA
 specifies asynchronous data.

mpe_Storage

`mpe_Storage` is an opaque handle for the media storage data structure. `mpe_Storage` is described in `mpeos_dvr.h` as follows:

```
typedef struct _mpe_StorageH { int unused1; } * mpe_Storage;
```

mpe_StorageHandle

`mpe_StorageHandle` represents a platform-independent handle or reference to a particular storage device. `mpe_StorageHandle` is defined in the `mpeos_storage.h` as follows:

```
typedef mpeos_Storage* mpe_StorageHandle;
```

mpeos_MediaVolume

`mpeos_MediaVolume` provides an opaque data type that represents a media storage volume. The internal representation of this data type `os_MediaVolumeInfo`, is specific to the platform dependent MPEOS implementation and is defined in `os_dvr.h`. `mpeos_MediaVolume` is described in the `mpeos_dvr.h` as follows:

```
typedef os_MediaVolumeInfo mpeos_MediaVolume;
```

Events

The following DVR events, which are defined in `mpeos_dvr.h`, are used by the DVR functions:

| | |
|---------------------------------------|--------------------------------------|
| <code>MPE_DVR_EVT_DEVICE_ERROR</code> | <code>MPE_DVR_EVT_SYSTEM_BUSY</code> |
| <code>MPE_DVR_EVT_SYSTEM_READY</code> | <code>mpe_DvrEvent</code> |

`MPE_DVR_EVT_DEVICE_ERROR`

`MPE_DVR_EVT_DEVICE_ERROR` indicates the DVR failed to initialize.
`MPE_DVR_EVT_DEVICE_ERROR` is described as follows:

```
#define MPE_DVR_EVT_DEVICE_ERROR 6
```

`MPE_DVR_EVT_SYSTEM_BUSY`

`MPE_DVR_EVT_SYSTEM_BUSY` indicates the DVR system is currently busy.
`MPE_DVR_EVT_SYSTEM_BUSY` is described as follows:

```
#define MPE_DVR_EVT_SYSTEM_BUSY 2
```

`MPE_DVR_EVT_SYSTEM_READY`

`MPE_DVR_EVT_SYSTEM_READY` indicates the DVR system is ready.
`MPE_DVR_EVT_SYSTEM_READY` is described as follows:

```
#define MPE_DVR_EVT_SYSTEM_READY 0
```

`mpe_DvrEvent` `mpe_DvrEvent` defines the events of the DVR functions. `mpe_DvrEvent` is described as follows:

```
typedef enum mpe_DvrEvent {
    MPE_DVR_EVT_OUT_OF_SPACE = 0x1000,
    MPE_DVR_EVT_END_OF_FILE,
    MPE_DVR_EVT_START_OF_FILE,
    MPE_DVR_EVT_CONVERSION_STOP,
    MPE_DVR_EVT_PLAYBACK_PIDCHANGE,
    MPE_DVR_EVT_SESSION_CLOSED,
} mpe_DvrEvent;
```

where

`MPE_DVR_EVT_OUT_OF_SPACE`

indicates the DVR is out of storage space.

`MPE_DVR_EVT_END_OF_FILE`

indicates the playback has reached the end of the file.

`MPE_DVR_EVT_START_OF_FILE`

indicates the playback has reached the start of the file.

`MPE_DVR_EVT_CONVERSION_STOP`

notifies the Java layer that a Time-Shift Buffer (TSB) conversion has stopped.

MPE_DVR_EVT_PLAYBACK_PIDCHANGE
indicates a PID change during a playback session.

MPE_DVR_EVT_SESSION_CLOSED
indicates a DVR session is complete.

Supported functions

The following DVR functions need to be supported:

- `mpeos_dvrFreeRecordingList`
releases the memory allocated for the recording list.
- `mpeos_dvrGet` gets generic recording information for the storage device.
- `mpeos_dvrGetLowPowerResumeTime`
gets the time it takes to resume from low-power state.
- `mpeos_dvrGetPlayScales`
gets all the supported play scales.
- `mpeos_dvrGetRecordingList`
gets a list of existing recording names.
- `mpeos_dvrGetSystemStatus`
gets DVR system status.
- `mpeos_dvrGetTrickMode`
gets the current trick mode.
- `mpeos_dvrIsDecodable`
determines whether the recording can be decoded.
- `mpeos_dvrIsDecryptable`
determines whether the recording can be decrypted.
- `mpeos_dvrMediaVolumeAddAlarm`
registers a free space alarm.
- `mpeos_dvrMediaVolumeDelete`
deletes a media storage volume.
- `mpeos_dvrMediaVolumeGetCount`
gets the number of media storage volumes on the device.
- `mpeos_dvrMediaVolumeGetInfo`
gets information about an attribute on the media storage volume.
- `mpeos_dvrMediaVolumeGetList`
gets a list of media storage volumes on the device.
- `mpeos_dvrMediaVolumeNew`
creates a media storage volume.
- `mpeos_dvrMediaVolumeRegisterQueue`
registers a queue to receive media volume related events.
- `mpeos_dvrMediaVolumeRemoveAlarm`
removes a free space alarm.
- `mpeos_dvrMediaVolumeSetInfo`
sets the media storage volume attributes.
- `mpeos_dvrPlaybackChangePids`
allows a PID change on an active playback session.

`mpeos_dvrPlaybackGetPids`
gets the current PID information.

`mpeos_dvrPlaybackGetTime`
gets the current playback time.

`mpeos_dvrPlaybackChangePids`
allows a PID change on an active playback session.

`mpeos_dvrPlaybackGetTime`
gets the current playback time.

`mpeos_dvrPlaybackSetTime`
moves the playback to a specific time location.

`mpeos_dvrPlaybackStop`
stops a playback.

`mpeos_dvrRecordingDelete`
deletes a recording.

`mpeos_dvrRecordingGet`
gets recording information.

`mpeos_dvrRecordingPlayStart`
starts a playback of an existing recording.

`mpeos_dvrResumeFromLowPower`
restores regular-power state.

`mpeos_dvrSetTrickMode`
controls the playback speed and direction.

`mpeos_dvrTsbBufferingChangePids`
allows a PID change on an active buffering session.

`mpeos_dvrTsbBufferingStart`
starts a buffering session into the time-shift buffer.

`mpeos_dvrTsbBufferingStop`
stops a buffering session from the time-shift buffer.

`mpeos_dvrTsbChangeDuration`
reallocates the buffer based on duration.

`mpeos_dvrTsbConvertChangePids`
changes to the current converted PIDs.

`mpeos_dvrTsbConvertStart`
converts a time-shift record into a file.

`mpeos_dvrTsbConvertStop`
stops converting a time-shift recording into a file.

`mpeos_dvrTsbDelete`
deletes a time-shift buffer.

`mpeos_dvrTsbGet`
gets information about the time-shift buffer.

mpeos_dvrTsbNew
allocates space for a future time-shift buffer.

mpeos_dvrTsbPlayStart
creates and starts a playback session from a time-shift
buffer.

mpeos_dvrFreeRecordingList

releases the memory allocated for the recording list.

syntax mpe_Error mpeos_dvrFreeRecordingList(void);

parameter(s) none

value returned If the call is successful, mpeos_dvrFreeRecordingList() should return MPE_DVR_ERR_NOERR. Otherwise, it should return the following error code:

MPE_DVR_ERR_NOT_IMPLEMENTED

indicates DVR functionality is not implemented.

description mpeos_dvrFreeRecordingList() should release the memory allocated for the recording list returned by mpeos_dvrGetRecordingList().

related function(s) mpeos_dvrGetRecordingList

mpeos_dvrGet

gets generic recording information for the storage device.

syntax `mpe_Error mpeos_dvrGet(mpe_DvrInfoParam param, void * input, void * output);`

parameter(s) *param* specify the DVR information parameter. `mpe_DvrInfoParam` is described in the *mpe_DvrInfoParam* section. Currently, the following values are supported for *param*:

`MPE_DVR_SUPPORTS_CROSS_MSV_TSB_CONVERT`

indicates whether the platform supports the ability to convert TSB content located in one media storage volume to a linear recording in different MSV.

`MPE_DVR_STORAGE_MEDIAFS_CAPACITY`

indicates the storage capacity for MEDIAFS content across all media storage volume.

`MPE_DVR_STORAGE_MEDIAFS_ALLOCATABLE_SPACE`

indicates the free MEDIAFS space on the storage device not yet reserved by any media storage volume.

`MPE_DVR_STORAGE_MEDIAFS_FREE_SPACE`

indicates the free MEDIAFS space on the storage device including any unused space reserved by media storage volume.

input

is an optional input pointer used to define storage identifiers. If NULL, the information is retrieved from the default internal storage device.

output

is an output pointer to the information requested by *param*.

value returned

If the call is successful, `mpeos_dvrGet()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`

indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`

indicates an error occurred at the operating-system level.

`MPE_DVR_ERR_DEVICE_ERR`

indicates a hardware device error.

description

`mpeos_dvrGet()` should get generic recording information (for example, available space) for a specified storage device .

related function(s)

`mpeos_dvrGetRecordingList`

`mpeos_dvrFreeRecordingList`

mpeos_dvrGetLowPowerResumeTime

gets the time it takes to resume from low-power state.

syntax `uint32_t mpeos_dvrGetLowPowerResumeTime(void);`

parameter(s) none

value returned none

description `mpeos_dvrGetLowPowerResumeTime()` should get the amount of time needed for the slowest device to make a recording and resume from a low-power state.

related function(s) `mpeos_dvrResumeFromLowPower`

mpeos_dvrGetPlayScales

gets all the supported play scales.

syntax mpe_Error mpeos_dvrGetPlayScales(
 mpe_Storage *storage*,
 float ** *playScales*,
 uint32_t * *num*);

| | | |
|---------------------|-------------------|--|
| parameter(s) | <i>storage</i> | specifies a handle for the media storage. mpeos_storageGetDeviceList() returns the list of available storage device handles. <i>mpe_Storage</i> is described in the <i>mpe_Storage</i> section. Currently, <i>storage</i> is ignored because storage devices are not supported. |
| | <i>playScales</i> | is an output pointer to the play scales array (floating-point values). |
| | <i>num</i> | is an output pointer to the number of entries in the play scales array. |

value returned If the call is successful, *mpeos_dvrGetPlayScales()* should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_INVALID_PARAM
 indicates an invalid parameter.

MPE_DVR_ERR_OS_FAILURE
 indicates an error occurred at the operating-system level.

MPE_DVR_ERR_DEVICE_ERR
 indicates a hardware device error.

description *mpeos_dvrGetPlayScales()* should get all the play scales supported for the platform (for example, -2.0, -.25, .25, 2.0, etc.).

related function(s) none

mpeos_dvrGetRecordingList

gets a list of existing recording names.

syntax `mpe_Error mpeos_dvrGetRecordingList(mpe_Storage storage, uint32_t * count, mpe_DvrString_t ** recordingNames);`

| | | |
|---------------------|------------------------|--|
| parameter(s) | <i>storage</i> | specifies a handle for the media storage. <code>mpe_Storage</code> is described in the <i>mpe_Storage</i> section. Currently, <i>storage</i> is ignored because storage devices are not supported. |
| | <i>count</i> | is an output pointer to the number of recording names in the list. |
| | <i>recording names</i> | is an output pointer to the array of strings representing recording names. <code>mpe_DvrString_t</code> is described in the <i>mpe_DvrString_t</i> section. |

value returned If the call is successful, `mpeos_dvrGetRecordingList()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`
indicates an error occurred at the operating-system level.

`MPE_DVR_ERR_DEVICE_ERR`
indicates a hardware device error.

description `mpeos_dvrGetRecordingList()` should get a list of existing recording names from the specified storage device.

related function(s) `mpeos_dvrFreeRecordingList`

mpeos_dvrGetSystemStatus

gets DVR system status.

syntax `mpe_Error mpeos_dvrGetSystemStatus(mpe_EventQueue queueId , void * act, uint32_t * status);`

parameter(s) `queueId` identifies the queue in which to send the events associated with `status`. `mpe_EventQueue` is described in the `mpe_EventQueue` section. Currently, the following events are supported:

`MPE_DVR_EVT_DEVICE_ERROR`
indicates the DVR failed to initialize.

`MPE_DVR_EVT_SYSTEM_BUSY`
indicates the DVR system is currently busy.

`MPE_DVR_EVT_SYSTEM_READY`
indicates the DVR system is ready.

`act` is an output pointer to the notification completion token. Currently, there is no ACT data associated with a DVR event.

`status` is an output pointer to the current DVR system status and a queue is created for further notification of DVR events related to the DVR status.

value returned If the call is successful, `mpeos_dvrGetSystemStatus()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_DEVICE_ERR`
indicates a hardware device error.

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`
indicates an error occurred at the operating-system level.

description `mpeos_dvrGetSystemStatus()` should monitor changes in the DVR system status. If the system status changes, an event is sent asynchronously to .

related function(s) none

mpeos_dvrGetTrickMode

gets the current trick mode.

syntax mpe_Error mpeos_dvrGetTrickMode(
 mpe_DvrPlayback *playback*,
 float * *mode*);

parameter(s) *playback* specifies the handle to the playback. *playback* is returned by a previous call to mpeos_dvrRecordingPlayStart() or mpeos_dvrTSBPlayStart(). *mpe_DvrPlayback* is described in the *mpe_DvrPlayback* section.

mode is an output pointer to the current trick mode. Trick mode values vary based on the operating system. *mode* is returned by a previous call to mpeos_dvrSetTrickMode().

value returned If the call is successful, mpeos_dvrGetTrickMode() should return MPE_DVR_ERR_NOERR. Otherwise, it should return the following error code:
MPE_DVR_ERR_INVALID_PARAM
indicates an invalid parameter.

description mpeos_dvrGetTrickMode() should get the current trick mode associated with a specified time-shift buffer or recording playback.

related function(s) mpeos_dvrSetTrickMode

mpeos_dvrIsDecodable

determines whether the recording can be decoded.

syntax mpe_Bool mpeos_dvrIsDecodable(char * *recName*);

parameter(s) *recName* specifies the name of the recording.

value returned If the recording can be decoded, `mpeos_dvrIsDecodable()` should return TRUE. If the recording cannot be decoded, `mpeos_dvrIsDecodable()` should return FALSE.

description `mpeos_dvrIsDecodable()` should determine whether the recording can be decoded. `mpeos_dvrTsbPlayStart()` and `mpeos_dvrRecordingPlayStart()` decode the recording before playback.

related function(s) none

mpeos_dvrIsDecryptable

determines whether the recording can be decrypted.

syntax mpe_Bool mpeos_dvrIsDecryptable(char * *recName*);

parameter(s) *recName* specifies the name of the recording.

value returned If the recording can be decrypted, mpeos_dvrIsDecryptable() returns TRUE. If the recording cannot be decrypted, mpeos_dvrIsDecryptable() returns FALSE.

description mpeos_dvrIsDecryptable() should determine whether the recording can be decrypted. mpeos_dvrTsbPlayStart() and mpeos_dvrRecordingPlayStart() decrypt the recording before playback.

related function(s) none

mpeos_dvrMediaVolumeAddAlarm

registers a free space alarm.

syntax mpe_Error mpeos_dvrMediaVolumeAddAlarm(
 mpe_MediaVolume *volume*,
 uint8_t *level*);

parameter(s) *volume* specifies the handle to the media storage volume to monitor for free space. *volume* is returned by a previous call the mpeos_dvrMediaVolumeNew(). *mpe_MediaVolume* is described in the *mpe_MediaVolume* section. Currently, the following event is supported:

MPE_MEDIA_VOL_EVT_FREE_SPACE_ALARM
 specifies the media storage volumes is low on free space.

level specifies an alarm to notify the caller when the media storage volume capacity, in bytes, gets to a specified level. *level* is a percentage of the specified media storage volume's remaining capacity. The valid range of values is 1-99.

value returned If the call is successful, mpeos_dvrMediaVolumeAddAlarm() should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_INVALID_PARAM
 indicates an invalid parameter.

MPE_DVR_ERR_NOT_ALLOWED
 indicates the operation is not allowed.

description mpeos_dvrMediaVolumeAddAlarm() should register a free space alarm for the specified media storage volume. When the storage volume reaches the specified lowest level of free space, mpeos_dvrMediaVolumeAddAlarm() generates an MPE_MEDIA_VOL_EVT_FREE_SPACE_ALARM event and posts it to the associated queue. The queue is created by a previous call to mpeos_dvrMediaVolumeRegisterQueue(). Multiple alarms may be registered for the same volume, but the level must be unique across all free space alarms for that volume.

related function(s) mpeos_dvrMediaVolumeRemoveAlarm

mpeos_dvrMediaVolumeDelete

deletes a media storage volume.

syntax mpe_Error mpeos_dvrMediaVolumeDelete(mpe_MediaVolume *volume*);

parameter(s) *volume* specifies the media storage volume to delete. *volume* is returned by a previous call to mpeos_dvrMediaVolumeNew(). *mpe_MediaVolume* is described in the *mpe_MediaVolume* section.

value returned If the call is successful, mpeos_dvrMediaVolumeDelete() should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_DEVICE_ERR
indicates a hardware device error.

MPE_DVR_ERR_NOT_ALLOWED
indicates the operation is not allowed.

description mpeos_dvrMediaVolumeDelete() should delete the media storage volume specified by *volume*.

related function(s) mpeos_dvrMediaVolumeNew

mpeos_dvrMediaVolumeGetCount

gets the number of media storage volumes on the device.

syntax mpe_Error mpeos_dvrMediaVolumeGetCount(
 mpe_StorageHandle *device*,
 uint32_t * *count*);

parameter(s) *device* specifies the storage device. *device* is returned by a previous call to mpeos_storageGetDeviceList(). *mpe_StorageHandle* is described in the *mpe_StorageHandle* section.

count is an output pointer to the number of media storage volumes found on the storage device.

value returned If the call is successful, mpeos_dvrMediaVolumeGetCount() should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_DEVICE_ERR
 indicates a hardware device error.

MPE_DVR_ERR_INVALID_PARAM
 indicates an invalid parameter.

MPE_DVR_ERR_UNSUPPORTED
 indicates the operation is not supported.

description mpeos_dvrMediaVolumeGetCount() should get the number of media storage volumes found on the specified storage device.

related function(s) none

mpeos_dvrMediaVolumeGetInfo

gets information about an attribute on the media storage volume.

syntax mpe_Error mpeos_dvrMediaVolumeGetInfo (
 mpe_MediaVolume *volume*,
 mpe_MediaVolumeInfoParam *param*,
 void * *output*);

| | | |
|---------------------|--------------------------------------|--|
| parameter(s) | <i>volume</i> | specifies the handle to the media storage volume to get information about. <i>volume</i> is returned by a previous call to the mpeos_dvrMediaVolumeNew(). <i>mpe_MediaVolume</i> is described in the <i>mpe_MediaVolume</i> section. |
| | <i>param</i> | specifies the parameter to query. <i>mpe_MediaVolumeInfoParam</i> is described in the <i>mpe_MediaVolumeInfoParam</i> section. Currently, the following values are defined for <i>param</i> : |
| | MPE_DVR_MEDIA_VOL_SIZE | specifies the size of the media storage volume. |
| | MPE_DVR_MEDIA_VOL_FREE_SPACE | gets the size of the free space remaining in the media storage volumes. |
| | MPE_DVR_MEDIA_VOL_PATH | gets the path to the media storage volumes. |
| | MPE_DVR_MEDIA_VOL_ALLOWED_LIST_COUNT | gets the number of media storage volumes in the list. |
| | MPE_DVR_MEDIA_VOL_ALLOWED_LIST | specifies the number of media storage volumes allowed on the list. |
| | <i>output</i> | is an output pointer to the resulting value. This buffer is allocated by the caller. |

value returned If the call is successful, mpeos_dvrMediaVolumeGetInfo() should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

| | |
|---------------------------|---|
| MPE_DVR_ERR_INVALID_PARAM | indicates an invalid parameter. |
| MPE_DVR_ERR_NOT_ALLOWED | indicates the operation is not allowed. |

description mpeos_dvrMediaVolumeGetInfo() should get information about an attribute for the specified media storage volume.

related function(s) mpeos_dvrMediaVolumeSetInfo

mpeos_dvrMediaVolumeGetList

gets a list of media storage volumes on the device.

syntax `mpe_Error mpeos_dvrMediaVolumeGetList(mpe_StorageHandle device, uint32_t * count, mpe_MediaVolume * volumes);`

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>device</i> | specifies the storage device. <i>device</i> is returned by a previous call to <code>mpeos_storageGetDeviceList()</code> . <code>mpe_StorageHandle</code> is described in the <i>mpe_StorageHandle</i> section. |
| | <i>count</i> | is an output pointer to the number of volume handles returned in the <i>volumes</i> array. |
| | <i>volumes</i> | is an output pointer to a list of media storage handles on the specified device. <code>mpe_MediaVolume</code> is described in the <i>mpe_MediaVolume</i> section. |

value returned If the call is successful, `mpeos_dvrMediaVolumeGetList()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

| | |
|--|--|
| <code>MPE_DVR_ERR_DEVICE_ERR</code> | indicates a hardware device error. |
| <code>MPE_DVR_ERR_INVALID_PARAM</code> | indicates an invalid parameter. |
| <code>MPE_DVR_ERR_BUF_TOO_SMALL</code> | indicates that the pre-allocated <i>volumes</i> buffer is not large enough to fit all of the native volume handles found on the storage device. Only as many volume handles as can fit in the buffer are returned. |

description `mpeos_dvrMediaVolumeGetList()` should get a list of media storage volumes on the specified storage device. `mpeos_dvrMediaVolumeNew()` automatically adds devices to the list and `mpeos_dvrMediaVolumeDelete()` removes devices from the list.

related function(s) `mpeos_dvrMediaVolumeNew`
`mpeos_dvrMediaVolumeRemoveAlarm`

mpeos_dvrMediaVolumeNew

creates a media storage volume.

syntax mpe_Error mpeos_dvrMediaVolumeNew(
 mpe_StorageHandle *device*,
 char * *path*,
 mpe_MediaVolume * *volume*);

| | | |
|-----------------------|---------------------------|--|
| parameter(s) | <i>device</i> | specifies the handle to the storage device in which to create the media storage volume. <i>device</i> is returned by a previous call to mpeos_storageGetDeviceList(). <i>mpe_StorageHandle</i> is described in the <i>mpe_StorageHandle</i> section. |
| | <i>path</i> | is an input pointer to the path of the new media storage volume. The full path of the volume is presented to applications via LogicalStorageVolume.getPath() implemented at the Java layer. This path must be unique across all media storage volumes on the specified storage device. This path must begin with the MPE_STORAGE_GPFS_PATH of the specified storage device to be considered valid. No other identifiers are needed to uniquely identify this volume. This path is also used as the identifier for the media storage volume by uniquely identifying the corresponding LogicalStorageVolume instance maintained at the Java layer and because the MPEOS DVR module implemented on platforms that support exposing media files on a media volume through java.io would need this information. |
| | <i>volumes</i> | is an output pointer to the new media storage handle on the specified device. <i>mpe_MediaVolume</i> is described in the <i>mpe_MediaVolume</i> section. |
| value returned | | If the call is successful, mpeos_dvrMediaVolumeNew() should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes: |
| | MPE_DVR_ERR_DEVICE_ERR | indicates a hardware device error. |
| | MPE_DVR_ERR_INVALID_PARAM | indicates an invalid parameter. |
| | MPE_DVR_ERR_NOT_ALLOWED | indicates the operation is not allowed. |
| | MPE_DVR_ERR_UNSUPPORTED | indicates the operation is not supported. |

description

`mpeos_dvrMediaVolumeNew()` should create a new media storage volume on a specified storage device. By default, the newly created media volume will have no minimum guaranteed size and may use as much space as is available on the storage device that is not already reserved by other media volumes on that device. To change the size of the media storage volume, call `mpe_dvrMediaVolumeSetInfo(MPE_DVR_MEDIA_VOL_SIZE)`.



NOTE: `mpeos_dvrMediaVolumeNew()` does not accept volume name, organization identifiers, application identifiers, or extended file access permissions as input. The reason is because they are not needed at this level. Each media storage volume created at the MPE level will have a corresponding `MediaStorageVolume` object at the Java layer which inherits the functionality of a `LogicalStorageVolume` object. The volume name, organization identifier, application identifier, and extended file access permissions associated with the `LSV` object at the Java layer are also implicitly applied to the corresponding media storage volume. However, this information is not completely hidden from the MPE layer because the volume name, organization identifier, and application identifier are encoded in the path specified by the caller and the Extended File Access Permissions (EFAP) can be obtained from the MPE operating system Information Technology File System (ITFS) module using this same path.

related function(s)

`mpeos_dvrMediaVolumeDelete`

mpeos_dvrMediaVolumeRegisterQueue

registers a queue to receive media volume related events.

syntax `mpe_Error mpeos_dvrMediaVolumeRegisterQueue(mpe_EventQueue queueId , void * act);`

parameter(s) `queueId` identifies the queue in which to send the events associated with media volume. `mpe_EventQueue` is described in the `mpe_EventQueue` section. Currently, the following event is supported:

`MPE_MEDIA_VOL_EVT_FREE_SPACE_ALARM`
specifies the media storage volume is low on free space.

`act` is an output pointer to the notification completion token. Currently, there is no ACT data associated with a DVR event.

value returned If the call is successful, `mpeos_dvrMediaVolumeRegisterQueue()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return the following error code:

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

description `mpeos_dvrMediaVolumeRegisterQueue()` should register a queue to receive media volume related events. Only one queue may be registered at a time. Subsequent calls should replace a previously registered queue.

related function(s) none

mpeos_dvrMediaVolumeRemoveAlarm

removes a free space alarm.

syntax mpe_Error mpeos_dvrMediaVolumeRemoveAlarm(
 mpe_MediaVolume *volume*,
 uint8_t *level*);

parameter(s) *volume* specifies the handle to the media storage volume. *volume* is returned by a previous call the mpeos_dvrMediaVolumeNew(). *mpe_MediaVolume* is described in the *mpe_MediaVolume* section.

level specifies the percentage set for the specified media storage volume. The valid range of values is 1-99.

value returned If the call is successful, mpeos_dvrMediaVolumeRemoveAlarm() should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_INVALID_PARAM
 indicates an invalid parameter.

description mpeos_dvrMediaVolumeRemoveAlarm() should unregisters a free space alarm for the specified media storage volume.

related function(s) mpeos_dvrMediaVolumeAddAlarm

mpeos_dvrMediaVolumeSetInfo

sets the media storage volume attributes.

syntax `mpe_Error mpeos_dvrMediaVolumeSetInfo(mpe_MediaVolume volume, mpe_MediaVolumeInfoParam param, void * value);`

| | | |
|---------------------|---|---|
| parameter(s) | <i>volume</i> | specifies the handle to the media storage volume. <i>volume</i> is returned by a previous call to the <code>mpeos_dvrMediaVolumeNew()</code> . <code>mpe_MediaVolume</code> is described in the <i>mpe_MediaVolume</i> section. |
| | <i>param</i> | specifies the parameter to set. <code>mpe_MediaVolumeInfoParam</code> is described in the <i>mpe_MediaVolumeInfoParam</i> section. Currently, the following values are defined for <i>param</i> : |
| | <code>MPE_DVR_MEDIA_VOL_SIZE</code> | specifies the size of the media storage volume. |
| | <code>MPE_DVR_MEDIA_VOL_FREE_SPACE</code> | gets the size of the free space remaining in the media storage volumes. |
| | <code>MPE_DVR_MEDIA_VOL_PATH</code> | gets the path to the media storage volumes. |
| | <code>MPE_DVR_MEDIA_VOL_ALLOWED_LIST_COUNT</code> | gets the number of media storage volumes in the list. |
| | <code>MPE_DVR_MEDIA_VOL_ALLOWED_LIST</code> | specifies the number of media storage volumes allowed on the list. |
| <i>value</i> | | is an output pointer to the new value of <code>MPE_DVR_MEDIA_VOL_FREE_SPACE</code> . <i>value</i> is implementation specific and depends on <code>MPE_DVR_STORAGE_MEDIAFS_ALLOCATABLE_SPACE</code> . |

value returned If the call is successful, `mpeos_dvrMediaVolumeSetInfo()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_DEVICE_ERR`
indicates a hardware device error.

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_NOT_ALLOWED`
indicates the operation is not allowed.

description mpeos_dvrMediaVolumeSetInfo() should set media storage volume attributes.

related function(s) mpeos_dvrMediaVolumeGetInfo

mpeos_dvrPlaybackChangePids

allows a PID change on an active playback session.

syntax `mpe_Error mpeos_dvrPlaybackChangePids(mpe_DvrPlayback playback, mpe_DvrPidTable pidTable);`

- parameter(s)** *playback* specifies the handle to the playback. This handle is used to control trick play and media positions. *playback* is returned by a previous call to `mpeos_dvrRecordingPlayStart()` or `mpeos_dvrTsbPlayStart()`. `mpe_DvrPlayback` is described in the *mpe_DvrPlayback* section.
- pidTable* is an input pointer the DVR PID table. `mpe_DvrPidTable` is described in the *mpe_DvrPidTable* section.

value returned If the call is successful, `mpeos_dvrPlaybackChangePids()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_UNSUPPORTED`
indicates the operation is not supported.

description `mpeos_dvrPlaybackChangePids()` should allow a PID change on an active playback session. PIDs are initially set by `mpeos_dvrRecordingPlayStart()`. If the stream has different audio PIDs (for example, English and Spanish), `mpeos_dvrPlaybackChangePids()` allows you to change audio PIDs.

related function(s) `mpeos_dvrTsbBufferingChangePids`

mpeos_dvrPlaybackGetPids

gets the current PID information.

syntax mpe_Error mpeos_dvrPlaybackGetPids(
 mpe_DvrPlayback *playback*,
 mpe_DvrPidTable * *pidTable*);

parameter(s) *playback* specifies the handle to an active playback. *playback* is

returned by a previous call to

`mpeos_dvrRecordingPlayStart()` or

`mpeos_dvrTSBPlayStart()`. `mpe_DvrPlayback` is described in

the *mpe_DvrPlayback* section.

pidTable is an output pointer the DVR PID table. `mpe_DvrPidTable` is described in the *mpe_DvrPidTable* section.

value returned If the call is successful, `mpeos_dvrPlaybackGetPids()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`

indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`

indicates an error occurred at the operating-system level.

description `mpeos_dvrPlaybackGetPids()` should get the current PID information.

related function(s) `mpeos_dvrPlaybackChangePids`

mpeos_dvrPlaybackGetTime

gets the current playback time.

syntax mpe_Error mpeos_dvrPlaybackGetTime(
 mpe_DvrPlayback *playback*,
 int64 * *mediaTime*);

parameter(s) *playback*

specifies the handle to the playback. This handle is used to control trick play and media positions. *playback* is returned by a previous call to `mpeos_dvrRecordingPlayStart()` or `mpeos_dvrTSBPlayStart()`. `mpe_DvrPlayback` is described in the *mpe_DvrPlayback* section.

mediaTime

is an output pointer to the current media time in nanoseconds. *mediaTime* is returned by a previous call to `mpeos_dvrSetTime()`.

value returned

If the call is successful, `mpeos_dvrPlaybackGetTime()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`

indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`

indicates an error occurred at the operating-system level.

description

`mpeos_dvrPlaybackGetTime()` should get the current playback time. This time should represent current position in the playback.

related function(s)

`mpeos_dvrPlaybackSetTime`

mpeos_dvrPlaybackSetTime

moves the playback to a specific time location.

syntax mpe_Error mpeos_dvrPlaybackSetTime(
 mpe_DvrPlayback *playback*,
 int64 *mediaTime*);

parameter(s) *playback* specifies the handle to the playback. This handle is used to control trick play and media positions. *playback* is returned by a previous call to `mpeos_dvrRecordingPlayStart()` or `mpeos_dvrTSBPlayStart()`. `mpe_DvrPlayback` is described in the *mpe_DvrPlayback* section.

mediaTime specifies the current position in nanoseconds in which to jump. If media time is equal to `MPE_POSITIVE_INFINITY`, the playback terminates and the JMF player jumps to the live broadcast.

value returned If the call is successful, `mpeos_dvrPlaybackSetTime()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`
 indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`
 indicates an error occurred at the operating-system level.

description `mpeos_dvrPlaybackSetTime()` should move the playback position to a specified time.

related function(s) `mpeos_dvrPlaybackGetPids`

mpeos_dvrPlaybackStop

stops a playback.

syntax mpe_Error mpeos_dvrPlaybackStop(mpe_DvrPlayback *playback*);

parameter(s) *playback* specifies the handle to the playback. This handle is used to control trick play and media positions. *playback* is returned by a previous call to mpeos_dvrRecordingPlayStart() or mpeos_dvrTsbPlayStart(). *mpe_DvrPlayback* is described in the *mpe_DvrPlayback* section.

value returned If the call is successful, mpeos_dvrPlaybackStop() should return MPE_DVR_ERR_NOERR. Otherwise, it should return the following error code:

MPE_DVR_ERR_INVALID_PARAM
indicates an invalid parameter.

description mpeos_dvrPlaybackStop() should stop the specified playback. An event should be sent to notify the caller that the playback state has changed. To indicate that a DVR playback session has stopped, the MPE_DVR_EVT_SESSION_CLOSED event is sent to the Java layer.

related function(s) mpeos_dvrRecordingPlayStart
mpeos_dvrTsbPlayStart

mpeos_dvrRecordingDelete

deletes a recording.

syntax mpe_Error mpeos_dvrRecordingDelete(char * *recordingName*);

parameter(s) *recordingName* is an input pointer to the unique name identifier.
recordingName is returned by a previous call to
mpeos_dvrRecordingPlayStart().

value returned If the call is successful, mpeos_dvrRecordingDelete() should return
MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error
codes:

MPE_DVR_ERR_INVALID_PARAM
indicates an invalid parameter.

MPE_DVR_ERR_NOT_ALLOWED
indicates the operation is not allowed.

MPE_DVR_ERR_OS_FAILURE
indicates an error occurred at the operating-system level.

description mpeos_dvrRecordingDelete() should remove the specified recorded
content from the disk drive. An active recording cannot be deleted until it
is in the MPE_DVR_STOPPED state.

related function(s) mpeos_dvrRecordingGet
mpeos_dvrPlaybackStop

mpeos_dvrRecordingGet

gets recording information.

syntax mpe_Error mpeos_dvrRecordingGet(
 char * *recordingName*,
 mpe_DvrInfoParam *param*,
 void * *output*);

parameter(s) *recordingName*

is an input pointer to the unique name identifier.
recordingName is returned by a previous call to
mpeos_dvrRecordingPlayStart().

param specifies the information parameter. Currently, the following values are supported for *param*:

MPE_DVR_MEDIA_TIME

indicates the position in the media stream.

MPE_DVR_RECORDING_SIZE

indicates the size of both the buffer and recording in bytes.

MPE_DVR_RECORDING_MSV

indicates the media storage volume where the recording is stored.

MPE_DVR_RECORDING_LENGTH

indicates the recording length in seconds.

MPE_DVR_PID_INFO

indicates the PID information.

MPE_DVR_PID_COUNT

indicates the number of PIDs.

output is an output pointer to the value for the parameter.

value returned

If the call is successful, *mpeos_dvrRecordingGet()* should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_DEVICE_ERR

indicates a hardware device error.

MPE_DVR_ERR_INVALID_PARAM

indicates an invalid parameter.

MPE_DVR_ERR_OS_FAILURE

indicates an error occurred at the operating-system level.

MPE_DVR_ERR_UNSUPPORTED

indicates the operation is not supported.

description mpeos_dvrRecordingGet() should get the recording information regardless of the state of the recording.

related function(s) mpeos_dvrRecordingPlayStart

mpeos_dvrRecordingPlayStart

starts a playback of an existing recording.

```
syntax mpe_Error mpeos_dvrRecordingPlayStart(
    char * recordingName,
    mpe_DispDevice videoDevice,
    mpe_DvrPidInfo * pids,
    uint32_t pidCount,
    int64 mediaTime,
    float playRate,
    mpe_EventQueue queueld ,
    void * act,
    mpe_DvrPlayback * playback );
```

parameter(s) *recordingName* is an input pointer to the unique name identifier.

videoDevice identifies the video device. *videoDevice* is returned by a previous call to `mpeos_dispGetDevices()`. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

pids is an input pointer to the array of PIDs to be played back. `mpe_DvrPidInfo` is described in the *mpe_DvrPidInfo* section.

pidCount specifies the number of PIDs in the array.

mediaTime the media time is in nanoseconds and can be any value between 0 (the beginning of the stream) and the recording length. *mediaTime* represents the starting place in the stream.

playRate specifies the rate at which the playback is started. Trick mode values vary based on the operating system.

queueld identifies the queue in which to send the DVR events. `mpe_EventQueue` is described in the *mpe_EventQueue* section. Currently, the following events are supported:

MPE_DVR_EVT_OUT_OF_SPACE
indicates the DVR is out of storage space.

MPE_DVR_EVT_END_OF_FILE
indicates the end of the file.

MPE_DVR_EVT_START_OF_FILE
indicates the start of file.

MPE_DVR_EVT_CONVERSION_STOP
notifies the Java layer that a Time-Shift Buffer (TSB) conversion has stopped.

MPE_DVR_EVT_PLAYBACK_PIDCHANGE
indicates a PID change during a playback session.

MPE_DVR_EVT_SESSION_CLOSED
indicates a DVR session is complete.

act is an output pointer to the notification completion token. Currently, there is no ACT data associated with a DVR event.

playback is an output pointer to the newly created playback handle. This handle is used to control trick play and media positions. `mpe_DvrPlayback` is described in the *mpe_DvrPlayback* section.

value returned If the call is successful, `mpeos_dvrRecordingPlayStart()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_DEVICE_ERR`
indicates a hardware device error.

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`
indicates an error occurred at the operating-system level.

description `mpeos_dvrRecordingPlayStart()` should playback an existing recording. The recording should start playing back at the normal rate (1.0) for the specified *mediaTime*.

related function(s) `mpeos_dvrPlaybackChangePids`
`mpeos_dvrPlaybackStop`
`mpeos_dvrSetTrickMode`

mpeos_dvrResumeFromLowPower

restores regular-power state.

syntax mpe_Error mpeos_dvrResumeFromLowPower(void);

parameter(s) none

value returned If the call is successful, mpeos_dvrResumeFromLowPower() should return MPE_DVR_ERR_NOERR. Otherwise, it should return the following error code:

MPE_DVR_ERR_OS_FAILURE

indicates an error occurred at the operating-system level.

description mpeos_dvrResumeFromLowPower() should cause all devices associated with making a recording resume to a regular-power state.

related function(s) mpeos_dvrGetLowPowerResumeTime

mpeos_dvrSetTrickMode

controls the playback speed and direction.

syntax `mpe_Error mpeos_dvrSetTrickMode(mpe_DvrPlayback playback, float mode, float * actualMode);`

| | | |
|---------------------|-------------------|--|
| parameter(s) | <i>playback</i> | specifies the handle to the playback. This handle is used to control trick play and media positions. <i>playback</i> is returned from a previous call to <code>mpeos_dvrRecordingPlayStart()</code> or <code>mpeos_dvrTSBPlayStart()</code> . <code>mpe_DvrPlayback</code> is described in the <i>mpe_DvrPlayback</i> section. |
| | <i>mode</i> | specifies the desired trick mode. <i>mode</i> can be returned from a previous call to <code>mpeos_dvrGetPlayScales()</code> . Trick mode values vary based on the operating system. |
| | <i>actualMode</i> | is an output pointer to the actual mode set by the DVR system. |

| | |
|-----------------------|--|
| value returned | If the call is successful, <code>mpeos_dvrSetTrickMode()</code> should return <code>MPE_DVR_ERR_NOERR</code> . Otherwise, it should return one of the following error codes: |
| | <code>MPE_DVR_ERR_INVALID_PARAM</code> indicates an invalid parameter. |
| | <code>MPE_DVR_ERR_OS_FAILURE</code> indicates an error occurred at the operating-system level. |

| | |
|--------------------|---|
| description | <code>mpeos_dvrSetTrickMode()</code> should set the playback speed and direction. Trick mode should determine the playback speed and direction, for example, 0.5 specifies forward at half the normal speed and -2.0 specifies rewind at twice the normal speed. Trick mode may not be supported by all operating systems. If trick mode is not supported, the closest supported trick mode should be set and returned. |
|--------------------|---|

| | |
|----------------------------|------------------------------------|
| related function(s) | <code>mpeos_dvrGetTrickMode</code> |
|----------------------------|------------------------------------|

mpeos_dvrTsbBufferingChangePids

allows a PID change on an active buffering session.

syntax

```
mpe_Error mpeos_dvrTsbBufferingChangePids(
    mpe_DvrBuffering tsbSession,
    mpe_DvrPidInfo * pids,
    uint32_t pidCount,
    int64 mediaTime );
```

| | | |
|---------------------|-------------------|---|
| parameter(s) | <i>tsbSession</i> | is an output pointer to the active DVR handle for a buffering session. <i>mpe_DvrBuffering</i> is described in the <i>mpe_DvrBuffering</i> section. |
| | <i>pids</i> | is an input pointer to the PID information. <i>mpe_DvrPidInfo</i> is described in the <i>mpe_DvrPidInfo</i> section. |
| | <i>pidCount</i> | specifies the number of PIDs in the array. |
| | <i>mediaTime</i> | specifies the position in the buffer where the PID change occurred. |

value returned If the call is successful, *mpeos_dvrTsbBufferingChangePids()* should return *MPE_DVR_ERR_NOERR*. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_INVALID_PARAM
indicates an invalid parameter.

MPE_DVR_ERR_UNSUPPORTED
indicates the operation is not supported.

description *mpeos_dvrTsbBufferingChangePids()* should allow a PID change on an active time-shift buffering session. PIDs are initially set by *mpeos_dvrTsbBufferingStart()*. If the stream has different audio PIDs (for example, English and Spanish), *mpeos_dvrTsbBufferingChangePids()* allows you to change audio PIDs.

related function(s) *mpeos_dvrPlaybackChangePids*

mpeos_dvrTsbBufferingStart

starts a buffering session into the time-shift buffer.

```
syntax mpe_Error mpeos_dvrTsbBufferingStart(
    uint32_t tunerID,
    mpe_DvrTsb buffer,
    mpe_DvrBitRate bitRate,
    mpe_EventQueue queueId ,
    void * act,
    uint32_t pidCount,
    mpe_DvrPidInfo * pids,
    mpe_DvrBuffering * tsbSession );
```

| | | |
|---------------------|--------------------------|--|
| parameter(s) | <i>tunerId</i> | specifies the tuner identifier. |
| | <i>buffer</i> | specifies the handle to a time-shift buffer. <i>buffer</i> is returned by a previous call to <code>mpeos_dvrTsbNew()</code> . <code>mpe_DvrTsb</code> is described in the <i>mpe_DvrTsb</i> section. |
| | <i>bitRate</i> | specifies the recording quality for the DVR functions. <code>mpe_DvrBitRate</code> is described in the <i>mpe_DvrBitRate</i> section. Currently, the following values are defined for <i>bitRate</i> : |
| | MPE_DVR_BITRATE_LOW | enables the lowest possible bit rate. |
| | MPE_DVR_BITRATE_MEDIUM | enables a moderate bit rate. |
| | MPE_DVR_BITRATE_HIGH | enables the highest bit rate for high-definition boxes. |
| | <i>queueId</i> | identifies the queue in which to send DVR events. <code>mpe_EventQueue</code> is described in the <i>mpe_EventQueue</i> section. Currently, the following events are supported: |
| | MPE_DVR_EVT_DEVICE_ERROR | indicates the DVR failed to initialize. |
| | MPE_DVR_EVT_SYSTEM_BUSY | indicates the DVR system is currently busy. |
| | MPE_DVR_EVT_SYSTEM_READY | indicates the DVR system is ready. |
| | <i>act</i> | is an output pointer to the notification completion token. Currently, there is no ACT data associated with a DVR event. |
| | <i>pidCount</i> | specifies the number of PIDs in the array. |
| | <i>pids</i> | is an input pointer to the PID information. <code>mpe_DvrPidInfo</code> is described in the <i>mpe_DvrPidInfo</i> section. |
| | <i>tsbSession</i> | is an output pointer to the active DVR handle for a buffering session. <code>mpe_DvrBuffering</code> is described in the <i>mpe_DvrBuffering</i> section. |

value returned If the call is successful, `mpeos_dvrTsbBufferingStart()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`

indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`

indicates an error occurred at the operating-system level.

`MPE_DVR_ERR_OUT_OF_SPACE`

indicates there is no more space on the hard disk drive.

description `mpeos_dvrTsbBufferingStart()` should start the recording of the current program in the given time-shift buffer specified by *buffer*. A buffering handle specified by *tsbSession* should be created and returned after the recording has started.

related function(s) `mpeos_dvrTsbBufferingStop`

mpeos_dvrTsbBufferingStop

stops a buffering session from the time-shift buffer.

syntax `mpe_Error mpeos_dvrTsbBufferingStop(mpe_DvrBuffering * tsbSession);`

parameter(s) `tsbSession` is an input pointer to the active DVR handle for a buffering session. `mpe_DvrBuffering` is described in the *mpe_DvrBuffering* section.

value returned If the call is successful, `mpeos_dvrTsbBufferingStop()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`
indicates an error occurred at the operating-system level.

description `mpeos_dvrTsbBufferingStop()` should stop the recording of the buffering associated with the time-shift buffer session specified by `tsbSession`. It also deletes the playback associated with the recording.

related function(s) `mpeos_dvrTsbBufferingStart`

mpeos_dvrTsbChangeDuration

reallocates the buffer based on duration.

syntax `mpe_Error mpeos_dvrTsbChangeDuration(
 mpe_DvrTsb buffer,
 int64 duration);`

parameter(s) *buffer* specifies the handle to a time-shift buffer. *buffer* is returned by a previous call to `mpeos_dvrTsbNew()`. `mpe_DvrTsb` is described in the *mpe_DvrTsb* section.
duration specifies the new duration in seconds.

value returned If the call is successful, `mpeos_dvrTsbChangeDuration()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

- `MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.
- `MPE_DVR_ERR_OS_FAILURE`
indicates an error occurred at the operating-system level.
- `MPE_DVR_ERR_UNSUPPORTED`
indicates the operation is not supported.

description `mpeos_dvrTsbChangeDuration()` should reallocate the buffer associated with the buffering session specified by *buffer*, based on the duration specified by *duration*. If the buffer is in use, existing content in the buffer should be preserved.

◆ **NOTE:** `mpeos_dvrTsbChangeDuration()` is not allow if the buffering session is in progress. If a call is attempted, `MPE_DVR_ERR_UNSUPPORTED` is returned.

related function(s) `mpeos_dvrTsbNew`

mpeos_dvrTsbConvertChangePids

changes to the current converted PIDs.

syntax mpe_Error mpeos_dvrTsbConvertChangePids(
 mpe_DvrConversion *conversion*,
 mpe_DvrPidInfo * *pids*,
 uint32_t *pidCount*);

| | | |
|---------------------|-------------------|---|
| parameter(s) | <i>conversion</i> | is an output pointer to the conversion session. <i>mpe_DvrConversion</i> is described in the <i>mpe_DvrConversion</i> section. |
| | <i>pids</i> | is an input pointer to the PID to convert. <i>mpe_DvrPidInfo</i> is described in the <i>mpe_DvrPidInfo</i> section. |
| | <i>pidCount</i> | specifies the number of PIDs in the array. |

value returned If the call is successful, *mpeos_dvrTsbConvertChangePids()* should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_INVALID_PARAM
indicates an invalid parameter.

MPE_DVR_ERR_OS_FAILURE
indicates an error occurred at the operating-system level.

description *mpeos_dvrTsbConvertChangePids()* should change to the current converted PIDs (set by *mpeos_dvrTsbConvertStart()*) during a conversion. If the platform does not support seamless PID changes, MPE_DVR_ERR_UNSUPPORTED is returned.

related function(s) *mpeos_dvrTsbConvertStart*

mpeos_dvrTsbConvertStart

converts a time-shift record into a file.

```
syntax mpe_Error mpeos_dvrTsbConvertStart(
    mpe_DvrTsb buffer,
    int64 * startTime,
    int64 duration,
    mpe_DvrBitRate bitRate,
    mpe_EventQueue queueId ,
    void * act,
    uint32_t pidTableCount,
    mpe_DvrPidTable * pidTable,
    mpe_DvrConversion * conversion,
    char * recordingName );
```

| | | |
|---------------------|----------------------|--|
| parameter(s) | <i>buffer</i> | specifies the handle to a time-shift buffer. <i>buffer</i> is returned by a previous call to <code>mpeos_dvrTsbNew()</code> . <code>mpe_DvrTsb</code> is described in the <i>mpe_DvrTsb</i> section. |
| | <i>startTime</i> | is an input pointer to the DVR system time in milliseconds of when to start the conversion. |
| | <i>duration</i> | specifies the length of the conversion in seconds. |
| | <i>bitRate</i> | specifies the recording quality for the DVR functions. Currently, the following values are defined for <i>bitRate</i> : |
| | | MPE_DVR_BITRATE_LOW enables the lowest possible bit rate. |
| | | MPE_DVR_BITRATE_MEDIUM enables a moderate bit rate. |
| | | MPE_DVR_BITRATE_HIGH enables the highest bit rate for high-definition boxes. |
| | <i>queueId</i> | identifies the queue in which to send the DVR events. <code>mpe_EventQueue</code> is described in the <i>mpe_EventQueue</i> section. Currently, the following events are supported: |
| | | MPE_DVR_EVT_CONVERSION_STOP notifies the caller the conversion is complete. |
| | | MPE_DVR_EVT_OUT_OF_SPACE notifies the caller the DVR does not have enough storage space to complete the conversion. |
| | <i>act</i> | is an output pointer to the notification completion token. Currently, there is no ACT data associated with a DVR event. |
| | <i>pidTableCount</i> | specifies the number of PID sets in the array. 0 if not supported. |

pidTable is an input and output pointer to the DVR PID table. If the conversion is started successfully, `mpe_dvrTsbConvertStart()` updates the table with recorded IDs. `mpe_DvrPidTable` is described in the *mpe_DvrPidTable* section.

conversion is an output pointer to the conversion session. `mpe_DvrConversion` is described in the *mpe_DvrConversion* section.

recordingName is an output pointer to the unique name identifier which represents the permanent recording.

value returned If the call is successful, `mpeos_dvrTsbConvertStart()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_DEVICE_ERR`
indicates a hardware device error.

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_NOT_ALLOWED`
indicates the operation is not allowed.

`MPE_DVR_ERR_OS_FAILURE`
indicates an error occurred at the operating-system level.

`MPE_DVR_ERR_OUT_OF_SPACE`
indicates there is no more space on the disk drive.

description `mpeos_dvrTsbConvertStart()` should convert a time-shift buffer into a file while the buffering is in progress. A new conversion handle specified by *conversion* should be returned, representing the converted time-shift recording.

related function(s) `mpeos_dvrTsbConvertStop`

mpeos_dvrTsbConvertStop

stops converting a time-shift recording into a file.

syntax mpe_Error mpeos_dvrTsbConvertStop(
 mpe_DvrConversion *conversion*,
 mpe_Bool *immediate*);

parameter(s) *conversion* is an output pointer to the conversion session.
mpe_DvrConversion is described in the *mpe_DvrConversion* section.

immediate determines an immediate stop. If *immediate* is TRUE, the conversion stops immediately. If *immediate* is FALSE, the conversion continues until it is complete. *mpe_Bool* is described in the *mpe_Bool* section.

value returned If the call is successful, *mpeos_dvrTsbConvertStop()* should return MPE_DVR_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_INVALID_PARAM
 indicates an invalid parameter.

MPE_DVR_ERR_OS_FAILURE
 indicates an error occurred at the operating-system level.

description *mpeos_dvrTsbConvertStop()* should stop an ongoing conversion of a time-shift buffer.

related function(s) *mpeos_dvrTsbConvertStart*

mpeos_dvrTsbDelete

deletes a time-shift buffer.

syntax `mpe_Error mpeos_dvrTsbDelete(mpe_dvrTsb buffer);`

parameter(s) `buffer` specifies the handle to a time-shift buffer. *buffer* is returned by a previous call to `mpeos_dvrTsbNew()`. `mpe_dvrTsb` is described in the *mpe_DvrTsb* section.

value returned If the call is successful, `mpeos_dvrTsbDelete()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`
indicates an error occurred at the operating-system level.

description `mpeos_dvrTsbDelete()` should delete the specified time-shift buffer and release all associated resources. On some platforms ongoing playbacks is stopped and the content of the buffer is lost.

related function(s) `mpeos_dvrTsbNew`
`mpeos_dvrTsbPlayStart`

mpeos_dvrTsbGet

gets information about the time-shift buffer.

syntax

```
mpe_Error mpeos_dvrTsbGet(
    mpe_dvrTsb buffer,
    mpe_DvrInfoParam param,
    int64 * time );
```

parameter(s) *buffer* specifies the handle to a time-shift buffer. *buffer* is returned by a previous call to `mpeos_dvrTsbNew()`. `mpe_dvrTsb` is described in the *mpe_DvrTsb* section.

param specifies the information parameter. Currently, the following values are supported for *param*:

MPE_DVR_TSB_START_TIME
indicates the beginning of the buffer in nanoseconds.

MPE_DVR_TSB_END_TIME
indicates the end of the buffer in nanoseconds.

time is an output pointer to the output value of the parameter.

value returned If the call is successful, `mpeos_dvrTsbGet()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

MPE_DVR_ERR_INVALID_PARAM
indicates an invalid parameter.

MPE_DVR_ERR_NOT_ALLOWED
indicates the operation is not allowed.

MPE_DVR_ERR_OS_FAILURE
indicates an error occurred at the operating-system level.

MPE_DVR_ERR_UNSUPPORTED
indicates the operation is not supported.

description `mpeos_dvrTsbGet()` should get information about the time-shift buffer currently buffering (for example, start time and end time). If there is no buffering in progress, *time* should return 0.

related function(s) `mpeos_dvrTsbPlayStart`

mpeos_dvrTsbNew

allocates space for a future time-shift buffer.

syntax `mpe_Error mpeos_dvrTsbNew(`
`mpe_Storage storage,`
`int64 duration,`
`mpe_DvrTsb * buffer);`

| | | |
|---------------------|-----------------|--|
| parameter(s) | <i>storage</i> | specifies a handle for the media storage. <i>storage</i> is returned from a previous call to storage manager. <i>mpe_Storage</i> is described in the <i>mpe_Storage</i> section. Currently, <i>storage</i> is ignored because storage devices are not supported. |
| | <i>duration</i> | specifies the duration of the time-shift buffer in seconds. |
| | <i>buffer</i> | is an output pointer to the handle of the new buffer. <i>mpe_DvrTsb</i> is described in the <i>mpe_DvrTsb</i> section. |

value returned If the call is successful, `mpeos_dvrTsbNew()` should return `MPE_DVR_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_DVR_ERR_INVALID_PARAM`
 indicates an invalid parameter.

`MPE_DVR_ERR_OS_FAILURE`
 indicates an error occurred at the operating-system level.

`MPE_DVR_ERR_OUT_OF_SPACE`
 indicates there is no more space on the disk drive.

description `mpeos_dvrTsbNew()` should allocate a new time-shift buffer of a specified size and get a handle. `mpeos_dvrTsbNew()` should not start the time-shift buffer recording, but pre-allocate space on the storage device for future use. It should secure a time-shift buffer is available at the time of recording.

related function(s) `mpeos_dvrTsbDelete`
`mpeos_dvrTsbGet`
`mpeos_dvrTsbPlayStart`

mpeos_dvrTsbPlayStart

creates and starts a playback session from a time-shift buffer.

```
syntax mpe_Error mpeos_dvrTsbPlayStart(
    mpe_dvrTsb buffer,
    mpe_DispDevice videoDevice,
    mpe_DvrPidInfo * pids,
    uint32_t pidCount,
    int64 mediaTime,
    float playRate,
    mpe_EventQueue * queueld ,
    void * act,
    mpe_DvrPlayback * playback );
```

| | | |
|---------------------|--------------------|---|
| parameter(s) | <i>buffer</i> | specifies the handle to a time-shift buffer. <i>buffer</i> is returned by a previous call to <code>mpeos_dvrTsbNew()</code> . <code>mpe_dvrTsb</code> is described in the <i>mpe_DvrTsb</i> section. |
| | <i>videoDevice</i> | identifies the video device. <i>videoDevice</i> is returned by a previous call to <code>mpeos_dispGetDevices()</code> . <code>mpe_DispDevice</code> is described in the <i>mpe_DispDevice</i> section. |
| | <i>pids</i> | is an input pointer to the PID information. <code>mpe_DvrPidInfo</code> is described in the <i>mpe_DvrPidInfo</i> section. |
| | <i>pidCount</i> | specifies the number of PIDs in the array. |
| | <i>mediaTime</i> | specifies where the playback started. |
| | <i>playRate</i> | specifies the rate at which the playback is started. |
| | <i>queueld</i> | is an input pointer to the queue in which to send the DVR events. <code>mpe_EventQueue</code> is described in the <i>mpe_EventQueue</i> section. Currently, the following events are supported: MPE_DVR_EVT_OUT_OF_SPACE indicates the DVR is out of storage space. MPE_DVR_EVT_END_OF_FILE indicates the end of the file. MPE_DVR_EVT_START_OF_FILE indicates the start of file. MPE_DVR_EVT_CONVERSION_STOP notifies the Java layer that a Time-Shift Buffer (TSB) conversion has stopped. MPE_DVR_EVT_PLAYBACK_PIDCHANGE indicates a PID change during a playback session. MPE_DVR_EVT_SESSION_CLOSED indicates a DVR session is complete. |
| | <i>act</i> | is an output pointer to the notification completion token. Currently, there is no ACT data associated with a DVR event. |

playback is an output pointer to the playback handle. This handle is used to control trick play and media positions.
mpe_DvrPlayback is described in the *mpe_DvrPlayback* section.

value returned If the call is successful, *mpeos_dvrTsbPlayStart()* should return *MPE_DVR_ERR_NOERR*. Otherwise, it should return one of the following error codes:

- MPE_DVR_ERR_DEVICE_ERR*
indicates a hardware device error.
- MPE_DVR_ERR_INVALID_PARAM*
indicates an invalid parameter.
- MPE_DVR_ERR_OS_FAILURE*
indicates an error occurred at the operating-system level.

description *mpeos_dvrTsbPlayStart()* should create and start a new DVR playback from a specified time-shift buffer. A new playback handle should be returned if the call is successful. The recording should start playing back at the normal rate (1.0). The returned handle can be used to control the speed, the direction, and the position of the playback in the stream.

related function(s) *mpeos_dvrTsbDelete*
mpeos_dvrTsbGet
mpeos_dvrTsbNew
mpeos_dvrTsbPlayStart

EVT Event API

Overview

The Multimedia Platform Environment (MPE) porting layer provides queue-based event primitives. The event Application Programming Interface (API) delivers to and retrieves from event queues consisting of a user-defined event identifier, optional opaque data values, and two optional `uint32_t` of payloads. The data values can be a simple primitive data type or can be utilized as a pointer to user-specific application data. Similarly, an application can use the optional `uint32_t` of payloads however it meets the application's requirements.

The event API provides access to basic inter-thread communication primitives that can be used to pass state information about actions and events occurring in the operating system (for example, user input events, media events, etc.) to threads needed to perform actions in response to those events. Or, the event APIs can be used to simply coordinate data and state information between threads.

Before reading this chapter, you should be:

- ◆ familiar with common operating system support mechanisms for events
- ◆ familiar with common operating system synchronization mechanisms

After reading this chapter, you should be able to port the event functionality within the OpenCable Application Platform (OCAP) stack

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the event functions:

| | |
|--------------------------|--------------------------|
| <code>MPE_EEVENT</code> | <code>MPE_EINVAL</code> |
| <code>MPE_EMUTEX</code> | <code>MPE_ENODATA</code> |
| <code>MPE_ENOMEM</code> | <code>MPEETIMEOUT</code> |
| <code>MPE_SUCCESS</code> | |

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

`MPE_EEVENT`

`MPE_EEVENT` indicates the creating, deleting, or processing of events failed. `MPE_EEVENT` is defined as follows:

```
#define MPE_EEVENT OS_EEVENT
```

`MPE_EINVAL`

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

`MPE_EMUTEX`

`MPE_EMUTEX` indicates creation, deletion, or acquisition of a mutex via the function failed. `MPE_EMUTEX` is defined as follows:

```
#define MPE_EMUTEX OS_EMUTEX
```

`MPE_ENODATA`

`MPE_ENODATA` indicates the associated resource, queue, etc. has no data available. `MPE_ENODATA` is defined as follows:

```
#define MPE_ENODATA OS_ENODATA
```

`MPE_ENOMEM`

`MPE_ENOMEM` indicates the function failed due to insufficient memory resource. `MPE_ENOMEM` is defined as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

`MPEETIMEOUT`

`MPEETIMEOUT` indicates the API has returned due to a time-out condition. `MPEETIMEOUT` is defined as follows:

```
#define MPEETIMEOUT OSETIMEOUT
```

`MPE_SUCCESS`

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_types.h` and `mpeos_event.h`, are used by the event functions:

| | |
|-----------------------------|------------------------|
| <code>mpe_Error</code> | <code>mpe_Event</code> |
| <code>mpe_EventQueue</code> | |

`mpe_Error`

`mpe_Error` specifies the type of error codes. `mpe_Error` and the MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_Event`

`mpe_Event` identifies an event. `mpe_Event` is defined in `mpeos_event.h` as follows:

```
typedef os_Event mpe_Event;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

Supported functions

The following event queue functions need to be supported:

`mpeos_eventQueueDelete`
deletes the specified event queue.

`mpeos_eventQueueNew`
creates a new event queue.

`mpeos_eventQueueNext`
gets the next event from the specified queue (non-blocking).

`mpeos_eventQueueSend`
posts the specified event to the specified queue.

`mpeos_eventQueueWaitNext`
gets the next event from the specified queue.

mpeos_eventQueueDelete

deletes the specified event queue.

syntax mpe_Error mpeos_eventQueueDelete(mpe_EventQueue *queueId*);

parameter(s) *queueId* identifies the queue to delete. *queueId* is returned by a previous call to mpeos_eventQueueNew(). mpe_EventQueue is described in the *mpe_EventQueue* section.

value returned If the call is successful, mpeos_eventQueueDelete() should delete the event queue specified by *queueId*, release any resources associated with its underlying implementation, and return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE EINVAL indicates *queueId* has an invalid value.

description mpeos_eventQueueDelete() should delete the event queue specified by *queueId*. mpeos_eventQueueDelete() should also release any resources associated with the underlying implementation.

related function(s) mpeos_eventQueueNew
mpeos_eventQueueNext
mpeos_eventQueueSend
mpeos_eventQueueWaitNext

mpeos_eventQueueNew

creates a new event queue.

syntax `mpe_Error mpeos_eventQueueNew(mpe_EventQueue * queueId, const char * queueName);`

parameter(s) `queueId` is an output pointer for returning the identifier of the new event queue. `mpe_EventQueue` is described in the `mpe_EventQueue` section.

`queueName` is an input pointer to a string identifier for the new event queue.

value returned If the call is successful, `mpeos_eventQueueNew()` should return the identifier of the new event queue specified by `queueId` and a value of `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EEVENT` indicates a new queue could not be created.

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE_EMUTEX` indicates creation of a mutex via the function failed.

`MPE_ENOMEM` indicates the function failed due to insufficient memory resources.

description `mpeos_eventQueueNew()` should create a new event queue and return the identifier in `queueId`.

related function(s) `mpeos_eventQueueDelete`
`mpeos_eventQueueNext`
`mpeos_eventQueueSend`
`mpeos_eventQueueWaitNext`

mpeos_eventQueueNext

gets the next event from the specified queue (non-blocking).

syntax `mpe_Error mpeos_eventQueueNext(mpe_EventQueue queueId, mpe_Event * eventId, void ** optionalEventData1, void ** optionalEventData2, uint32_t * optionalEventData3);`

parameter(s) `queueId` identifies the target event queue. `queueId` is returned by a previous call to `mpeos_eventQueueNew()`. `mpe_EventQueue` is described in the *mpe_EventQueue* section.

`eventId` is an output pointer for returning the next event in the queue. `mpe_Event` is described in the *mpe_Event* section.

`optionalEventData1` is an output pointer for returning optional event data (for example, the `queueId`) to be delivered along with the specified event.

`optionalEventData2` is an output pointer for returning optional event data (for example, the Asynchronous Completion Token (ACT)) to be delivered along with the specified event.

`optionalEventData3` is an output pointer for returning optional event data (for example, addition application data) to be delivered along with the specified event.

value returned If the call is successful, `mpeos_eventQueueNext()` should return the next event specified by `eventId`, any optional data, and a value of `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EEVENT` indicates a new queue could not be created.

`MPE EINVAL` indicates `queueId` has an invalid value.

`MPE_ENODATA` indicates no events are available in the queue.

description `mpeos_eventQueueNext()` should get the next event from the queue specified by `queueId` (non-blocking) and any optional event data specified by `optionalEventData1`, `optionalEventData2`, and `optionalEventData3`.

related function(s) `mpeos_eventQueueDelete`
`mpeos_eventQueueNew`
`mpeos_eventQueueSend`
`mpeos_eventQueueWaitNext`

mpeos_eventQueueSend

posts the specified event to the specified queue.

syntax `mpe_Error mpeos_eventQueueSend(mpe_EventQueue queueId, mpe_Event eventId, void * optionalEventData, void * optionalEventData2, uint32_t optionalEventData3);`

parameter(s) `queueId` identifies the target event queue. `queueId` is returned by a previous call to `mpeos_eventQueueNew()`. `mpe_EventQueue` is described in the *mpe_EventQueue* section.

`eventId` identifies the event to send to the queue. `mpe_Event` is described in the *mpe_Event* section.

`optionalEventData1` is an output pointer for returning optional event data (for example, the `queueId`) to be delivered along with the specified event.

`optionalEventData2` is an output pointer for returning optional event data (for example, the Asynchronous Completion Token (ACT)) to be delivered along with the specified event.

`optionalEventData3` is an output pointer for returning optional event data (for example, addition application data) to be delivered along with the specified event.

value returned If the call is successful, `mpeos_eventQueueSend()` should post the event in `eventId` to the event queue specified by `queueId`, any optional event data, and return a value of `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EEVENT` indicates `mpeos_eventQueueSend()` could not send the event due to a target-specific error condition.

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_eventQueueSend()` should post the event specified by `eventId` and any optional event data specified by `optionalEventData1`, `optionalEventData2`, and `optionalEventData3`.

related function(s) `mpeos_eventQueueDelete`
`mpeos_eventQueueNew`
`mpeos_eventQueueNext`
`mpeos_eventQueueWaitNext`

mpeos_eventQueueWaitNext

gets the next event from the specified queue.

syntax

```
mpe_Error mpeos_eventQueueWaitNext(
    mpe_EventQueue queueId,
    mpe_Event * eventId,
    void ** optionalEventData1,
    void ** optionalEventData2,
    uint32_t * optionalEventData3,
    uint32_t timeOut );
```

| | | |
|---------------------|---------------------------|---|
| parameter(s) | <i>queueId</i> | identifies the target event queue. <i>queueId</i> is returned by a previous call to <code>mpeos_eventQueueNew()</code> . <code>mpe_EventQueue</code> is described in the <i>mpe_EventQueue</i> section. |
| | <i>eventId</i> | is an output pointer for returning the next event in the queue. <code>mpe_Event</code> is described in the <i>mpe_Event</i> section. |
| | <i>optionalEventData1</i> | is an output pointer for returning optional event data (for example, the <i>queueId</i>) to be delivered along with the specified event. |
| | <i>optionalEventData2</i> | is an output pointer for returning optional event data (for example, the Asynchronous Completion Token (ACT)) to be delivered along with the specified event. |
| | <i>optionalEventData3</i> | is an output pointer for returning optional event data (for example, addition application data) to be delivered along the specified event. |
| | <i>timeOut</i> | specifies the maximum timeout period in milliseconds to wait for the next event. If <i>timeOut</i> is 0, the call should wait indefinitely. |

value returned If the call is successful, `mpeos_eventQueueWaitNext()` should get the next event specified by *eventId*, any optional data, and a value of `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

- `MPE_EEVENT` indicates `mpeos_eventQueueWaitNext()` could not send the next event due to a target-specific error condition.
- `MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.
- `MPEETIMEOUT` indicates the *timeOut* period expired while waiting for the next event.

description mpeos_eventQueueWaitNext() should get the next event from the event queue specified by *queueId* and any optional event data specified by *optionalEventData1*, *optionalEventData2*, and *optionalEventData3*. If no events are queued, mpeos_eventQueueWaitNext() should wait for the number of milliseconds specified by *timeOut* for the next event before returning. If *timeOut* is 0, the call to mpeos_eventQueueWaitNext() should wait indefinitely for the next event.

related function(s) mpeos_eventQueueDelete
mpeos_eventQueueNew
mpeos_eventQueueNext
mpeos_eventQueueSend

FIL File-System API

Overview

The file-system Application Programming Interface (API) provides a portable abstraction layer for the underlying file systems in the target system, such as hard-drives, object carousels, Read-Only Memory (ROM) file-systems, etc. The common Multimedia Platform Environment (MPE) layer of the file-system API provides common functionality for the underlying Multimedia Platform Environment Operating System (MPEOS) and file-system specific interfaces, including managing the device mount-points into the global file-system hierarchy (for example, provides support for the file \romfs\file1.txt to go to the Read-Only Memory File System (ROMFS) MPEOS file-system interface code to look for the file file1.txt). The MPEOS file-system interfaces should not be directly called by either Java Virtual Machine (JVM) or other MPE/MPEOS APIs as the MPE layer provides a portable abstraction to all of the file systems available on a system.

Before reading this chapter, you should be:

- ◆ familiar with file systems in general
- ◆ knowledgeable of the functions for any file systems provided natively by the operating system
- ◆ knowledgeable about the semantics of the `java.io` and `org.dvb.dsmcc` packages

After reading this chapter, you should be able to create a driver allowing a new file system to be mounted within the MPE file system

Porting note

The OCAP stack contains an object carousel implementation and there are several functions which exist only to support the object carousel. Porters are not expected to implement these functions, they are documented only for completeness. The object carousel functions are:

| | |
|---|------------------------------------|
| <code>mpeos_fileAddDII()</code> | <code>mpeos_filePrefetch()</code> |
| <code>mpeos_filePrefetchModule()</code> | <code>mpeos_steamClose()</code> |
| <code>mpeos_streamGetNumTaps()</code> | <code>mpeos_streamOpen()</code> |
| <code>mpeos_steamReadEvent()</code> | <code>mpeos_streamReadTap()</code> |

Definitions

The file-system API needs definitions in the following categories:

- ◆ *File status definitions*
- ◆ *Open attribute definitions*
- ◆ *Operating-system specific definitions*
- ◆ *Permission definitions*
- ◆ *Priority definitions*
- ◆ *Seek definitions*
- ◆ *Type-support definitions*

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

File status definitions

The following definitions, which are defined in `mpe_file.h`, are used for status support:

| | |
|-------------------------------------|----------------------------------|
| <code>MPE_FS_STAT_CREATEDATE</code> | <code>MPE_FS_STAT_EXPDATE</code> |
| <code>MPE_FS_STAT_ISKNOWN</code> | <code>MPE_FS_STAT_PERM</code> |
| <code>MPE_FS_STAT_PRIOR</code> | <code>MPE_FS_STAT_SIZE</code> |
| <code>MPE_FS_STAT_SOURCEID</code> | <code>MPE_FS_STAT_TYPE</code> |

`MPE_FS_STAT_CREATEDATE`

`MPE_FS_STAT_CREATEDATE` gets the date the file was created.
`MPE_FS_STAT_CREATEDATE` is defined as follows:

```
#define MPE_FS_STAT_CREATEDATE (7)
```

`MPE_FS_STAT_EXPDATE`

`MPE_FS_STAT_EXPDATE` gets or sets the expiration date.
`MPE_FS_STAT_EXPDATE` is defined as follows:

```
#define MPE_FS_STAT_EXPDATE (9)
```

MPE_FS_STAT_ISKNOWN

MPE_FS_STAT_ISKNOWN gets a TRUE or FALSE value for the object. Objects can be files, directories, streams, and event streams. The object can be known by the implementation without being read into memory. For example, a file may be known when the parent directory is read indicating the file exists even though the file has not been read.

-
- ◆ **NOTE:** The object carousel uses MPE_FS_STAT_ISKNOWN to confirm the path to the directory.
-

If the directory for file is known, TRUE is returned. Otherwise, FALSE is returned. MPE_FS_STAT_ISKNOWN is defined as follows:

```
#define MPE_FS_STAT_ISKNOWN (6)
```

MPE_FS_STAT_MODDATE

MPE_FS_STAT_MODDATE gets the last date the file was modified.

MPE_FS_STAT_MODDATE is defined as follows:

```
#define MPE_FS_STAT_MODDATE (8)
```

MPE_FS_STAT_ORG_ACCESS

MPE_FS_STAT_ORG_ACCESS gets and sets the list of organization IDs that have read and/or write access. MPE_FS_STAT_ORG_ACCESS is defined as follows:

```
#define MPE_FS_STAT_ORG_ACCESS (21)
```

MPE_FS_STAT_PERM

MPE_FS_STAT_PERM gets and sets the permission for a Unix style file or directory. MPE_FS_STAT_PERM is defined as follows:

```
#define MPE_FS_STAT_PERM (2)
```

MPE_FS_STAT_PRIOR

MPE_FS_STAT_PRIOR gets or sets the priority of the file. MPE_FS_STAT_PRIOR is defined as follows:

```
#define MPE_FS_STAT_PRIOR (4)
```

MPE_FS_STAT_SIZE

MPE_FS_STAT_SIZE gets or sets the size of the file. For a file object, it specifies the number of bytes in the file. MPE_FS_STAT_SIZE is defined as follows:

```
#define MPE_FS_STAT_SIZE (1)
```

MPE_FS_STAT_TYPE

MPE_FS_STAT_TYPE gets the type of file. MPE_FS_STAT_TYPE is defined in mpe_file.h as follows:

```
#define MPE_FS_STAT_TYPE (3)
```

**Broadcast file
system file-status
definitions**

The following definitions, which are defined in `mpe_file.h`, are used for setting and retrieving file or directory status attributes:

| | |
|-----------------------------|-----------------------|
| MPE_FS_STAT_CONNECTIONAVAIL | MPE_FS_STAT_MOUNTPATH |
| MPE_FS_STAT_SOURCEID | |

MPE_FS_STAT_CONNECTIONAVAIL

`MPE_FS_STAT_CONNECTIONAVAIL` gets a TRUE or FALSE value based on the connection status for the specified network directory. If the network connection is available for the directory, TRUE is returned. Otherwise, FALSE is returned. the network connection is available for the specified directory. `MPE_FS_STAT_CONNECTIONAVAIL` is defined as follows:

```
#define MPE_FS_STAT_CONNECTIONAVAIL (102)
```

MPE_FS_STAT_MOUNTPATH

`MPE_FS_STAT_MOUNTPATH` gets the absolute system mount paths for the indicated object path. `MPE_FS_STAT_MOUNTPATH` is defined as follows:

```
#define MPE_FS_STAT_MOUNTPATH (101)
```

MPE_FS_STAT_SOURCEID

`MPE_FS_STAT_SOURCEID` gets the source identification of the file location. `MPE_FS_STAT_SOURCEID` is defined as follows:

```
#define MPE_FS_STAT_SOURCEID (12)
```

Open attribute definitions

The open attributes definitions are used by `mpeos_fileOpen()`. The following definitions, which are defined in `mpe_file.h`, are used to specify how the file is opened:

| | |
|--------------------------------------|-------------------------------------|
| <code>MPE_FS_OPEN_APPEND</code> | <code>MPE_FS_OPEN_CAN_CREATE</code> |
| <code>MPE_FS_OPEN_MUST_CREATE</code> | <code>MPE_FS_OPEN_READ</code> |
| <code>MPE_FS_OPEN_READWRITE</code> | <code>MPE_FS_OPEN_TRUNCATE</code> |
| <code>MPE_FS_OPEN_WRITE</code> | |

- ◆ **NOTE:** These definitions can be used in combinations by bitwise-ORing and are used only by the `openMode` parameter in `mpeos_fileOpen()`.

`MPE_FS_OPEN_APPEND`

`MPE_FS_OPEN_APPEND` opens the file so that writing begins at the end of the file. `MPE_FS_OPEN_APPEND` is used only in conjunction with `MPE_FS_OPEN_WRITE` or `MPE_FS_OPEN_READWRITE`. `MPE_FS_OPEN_APPEND` is defined as follows:

```
#define MPE_FS_OPEN_APPEND (0x0040)
```

`MPE_FS_OPEN_CAN_CREATE`

`MPE_FS_OPEN_CAN_CREATE` creates a file if necessary. The file must also be opened for writing. `MPE_FS_OPEN_CAN_CREATE` is defined as follows:

```
#define MPE_FS_OPEN_CAN_CREATE (0x0008)
```

`MPE_FS_OPEN_MUST_CREATE`

`MPE_FS_OPEN_MUST_CREATE` creates a file. The file must also be opened for writing. `MPE_FS_OPEN_MUST_CREATE` is defined as follows:

```
#define MPE_FS_OPEN_MUST_CREATE (0x0010)
```

`MPE_FS_OPEN_READ`

`MPE_FS_OPEN_READ` opens the file for read-only. Reads take place at the beginning of the file. `MPE_FS_OPEN_READ` is defined as follows:

```
#define MPE_FS_OPEN_READ (0x0001)
```

`MPE_FS_OPEN_READWRITE`

`MPE_FS_OPEN_READWRITE` opens a file for reading and writing. `MPE_FS_OPEN_READWRITE` is defined as follows:

```
#define MPE_FS_OPEN_READWRITE (0x0004)
```

`MPE_FS_OPEN_TRUNCATE`

`MPE_FS_OPEN_TRUNCATE` opens the file and deletes any existing content and writes to the beginning of the file. The file must also be opened for writing. `MPE_FS_OPEN_TRUNCATE` is defined as follows:

```
#define MPE_FS_OPEN_TRUNCATE (0x0020)
```

MPE_FS_OPEN_WRITE

MPE_FS_OPEN_WRITE opens a file for writing. MPE_FS_OPEN_WRITE is defined as follows:

```
#define MPE_FS_OPEN_WRITE (0x0002)
```

Operating-system specific definitions

The following definition, which is defined in `mpe_file.h`, is used for operating-system specific support:

```
MPE_FS_MAX_PATH
```

MPE_FS_MAX_PATH

MPE_FS_MAX_PATH specifies the maximum number of characters in the pathname. MPE_FS_MAX_PATH is defined as follows:

```
#define MPE_FS_MAX_PATH OS_FS_MAX_PATH
```

Permission definitions

The permission definitions are used only by `mpeos_getFStat()`, `mpeos_getStat()`, `mpeos_setFStat()`, and `mpeos_setStat()` when the `mode` parameter is `MPE_FS_STAT_PERM`. The following definitions, which are defined in `mpe_file.h`, are used for permission support:

| | |
|--------------------------------------|-------------------------------------|
| <code>MPE_FS_PERM_GROUP_READ</code> | <code>MPE_FS_PERM_OWNER_READ</code> |
| <code>MPE_FS_PERM_OWNER_WRITE</code> | <code>MPE_FS_PERM_WORLD_READ</code> |
| <code>MPE_FS_PERM_WORLD_WRITE</code> | |

- ◆ **NOTE:** These definitions can be used in combinations by bitwise-ORing.

`MPE_FS_PERM_GROUP_EXEC`

`MPE_FS_PERM_GROUP_EXEC` object has all run permissions.
`MPE_FS_PERM_WORLD_EXEC` is defined as follows:

```
#define MPE_FS_PERM_GROUP_EXEC (0x00020)
```

`MPE_FS_PERM_GROUP_READ`

`MPE_FS_PERM_GROUP_READ` object has organization read permissions.
`MPE_FS_PERM_GROUP_READ` is defined as follows:

```
#define MPE_FS_PERM_GROUP_READ (0x00008)
```

`MPE_FS_PERM_GROUP_WRITE`

`MPE_FS_PERM_GROUP_WRITE` object has organization write permissions.
`MPE_FS_PERM_GROUP_WRITE` is defined as follows:

```
#define MPE_FS_PERM_GROUP_WRITE (0x00010)
```

`MPE_FS_PERM_OWNER_EXEC`

`MPE_FS_PERM_OWNER_EXEC` object has all application permissions.
`MPE_FS_PERM_OWNER_EXEC` is defined as follows:

```
#define MPE_FS_PERM_OWNER_EXEC (0x00004)
```

`MPE_FS_PERM_OWNER_READ`

`MPE_FS_PERM_OWNER_READ` object has application read permissions.
`MPE_FS_PERM_OWNER_READ` is defined as follows:

```
#define MPE_FS_PERM_OWNER_READ (0x00001)
```

`MPE_FS_PERM_OWNER_WRITE`

`MPE_FS_PERM_OWNER_WRITE` object has application write permissions.
`MPE_FS_PERM_OWNER_WRITE` is defined as follows:

```
#define MPE_FS_PERM_OWNER_WRITE (0x00002)
```

`MPE_FS_PERM_WORLD_EXEC`

`MPE_FS_PERM_WORLD_EXEC` object has all run permissions.
`MPE_FS_PERM_WORLD_EXEC` is defined as follows:

```
#define MPE_FS_PERM_WORLD_EXEC (0x00100)
```

MPE_FS_PERM_WORLD_READ

MPE_FS_PERM_WORLD_READ object has all read permissions.
MPE_FS_PERM_WORLD_READ is defined as follows:

```
#define MPE_FS_PERM_WORLD_READ (0x00040)
```

MPE_FS_PERM_WORLD_WRITE

MPE_FS_PERM_WORLD_WRITE object has all write permissions.
MPE_FS_PERM_WORLD_WRITE is defined as follows:

```
#define MPE_FS_PERM_WORLD_WRITE (0x00080)
```

Priority definitions

The priority definitions are used only by `mpeos_getFStat()`, `mpeos_getStat()`, `mpeos_setFStat()`, and `mpeos_setStat()` when the `mode` parameter is `MPE_FS_STAT_PRIOR`. The following definitions, which are defined in `mpe_file.h`, are used for priority support:

`MPE_FS_PRIOR_HI`
`MPE_FS_PRIOR_MED`

`MPE_FS_PRIOR_LOW`

MPE_FS_PRIOR_HI `MPE_FS_PRIOR_HI` is a high-priority file. `MPE_FS_PRIOR_HI` is defined as follows:

```
#define MPE_FS_PRIOR_HI (3)
```

MPE_FS_PRIOR_LOW

`MPE_FS_PRIOR_LOW` is a low-priority file. `MPE_FS_PRIOR_LOW` is defined as follows:

```
#define MPE_FS_PRIOR_LOW (1)
```

MPE_FS_PRIOR_MED

`MPE_FS_PRIOR_MED` is a medium-priority file. `MPE_FS_PRIOR_MED` is defined as follows:

```
#define MPE_FS_PRIOR_MED (2)
```

Seek definitions

The seek definitions are used only by `mpeos_fileSeek()` as the `seekMode` parameter. The following definitions, which are defined in `mpe_file.h`, are used for seek support:

MPE_FS_SEEK_CUR
MPE_FS_SEEK_SET

MPE_FS_SEEK_END

MPE_FS_SEEK_CUR

MPE_FS_SEEK_CUR seeks from the current position of the file.
MPE_FS_SEEK_CUR is defined as follows:

```
#define MPE_FS_SEEK_CUR (2)
```

MPE_FS_SEEK_END

MPE_FS_SEEK_END seeks from the end of the file. MPE_FS_SEEK_END is defined as follows:

```
#define MPE_FS_SEEK_END (3)
```

MPE_FS_SEEK_SET

MPE_FS_SEEK_SET seeks from the beginning of the file. MPE_FS_SEEK_SET is defined as follows:

```
#define MPE_FS_SEEK_SET (1)
```

Type-support definitions

The type-support definitions are used only by `mpeos_getFStat()` and `mpeos_getStat()` when the `mode` parameter is `MPE_FS_STAT_TYPE`. The following definitions, which are defined in `mpe_file.h`, are used for type support:

| | |
|--------------------------------------|----------------------------------|
| <code>MPE_FS_TYPE_DIR</code> | <code>MPE_FS_TYPE_FILE</code> |
| <code>MPE_FS_TYPE_OTHER</code> | <code>MPE_FS_TYPE_STREAM</code> |
| <code>MPE_FS_TYPE_STREAMEVENT</code> | <code>MPE_FS_TYPE_UNKNOWN</code> |

`MPE_FS_TYPE_DIR`

`MPE_FS_TYPE_DIR` indicates the object type is a directory. `MPE_FS_TYPE_DIR` is defined as follows:

```
#define MPE_FS_TYPE_DIR (0x0002)
```

`MPE_FS_TYPE_FILE`

`MPE_FS_TYPE_FILE` indicates the object type is a file. `MPE_FS_TYPE_FILE` is defined as follows:

```
#define MPE_FS_TYPE_FILE (0x0001)
```

`MPE_FS_TYPE_OTHER`

`MPE_FS_TYPE_OTHER` indicates the object type is known.

`MPE_FS_TYPE_OTHER` allows additional file systems to support other object types. It is not used in the current OCAP stack. `MPE_FS_TYPE_OTHER` is defined as follows:

```
#define MPE_FS_TYPE_OTHER (0xffff)
```

`MPE_FS_TYPE_STREAM`

`MPE_FS_TYPE_STREAM` indicates the object type is stream.

`MPE_FS_TYPE_STREAM` is defined as follows:

```
#define MPE_FS_TYPE_STREAM (0x0003)
```

`MPE_FS_TYPE_STREAMEVENT`

`MPE_FS_TYPE_STREAMEVENT` indicates the object type is stream event.

`MPE_FS_TYPE_STREAMEVENT` is defined as follows:

```
#define MPE_FS_TYPE_STREAMEVENT (0x0004)
```

`MPE_FS_TYPE_UNKNOWN`

`MPE_FS_TYPE_UNKNOWN` indicates the object type is unknown.

`MPE_FS_TYPE_UNKNOWN` is defined as follows:

```
#define MPE_FS_TYPE_UNKNOWN (0x0000)
```

Error codes

The following error codes, which are defined in `mpe_file.h`, are used by the file-system functions:

| | |
|---|--|
| <code>MPE_FS_ERROR_ALREADY_EXISTS</code> | <code>MPE_FS_ERROR_DEVICE_FAILURE</code> |
| <code>MPE_FS_ERROR_EOF</code> | <code>MPE_FS_ERROR_EVENTS</code> |
| <code>MPE_FS_ERROR_FAILURE</code> | |
| <code>MPE_FS_ERROR_INVALID_PARAMETER</code> | <code>MPE_FS_ERROR_INVALID_STATE</code> |
| <code>MPE_FS_ERROR_NOT_FOUND</code> | <code>MPE_FS_ERROR_NO_MOUNT</code> |
| <code>MPE_FS_ERROR NOTHING_TO_ABORT</code> | <code>MPE_FS_ERROR_OUT_OF_MEM</code> |
| <code>MPE_FS_ERROR_READ_ONLY</code> | <code>MPE_FS_ERROR_SUCCESS</code> |
| <code>MPE_FS_ERROR_TIMEOUT</code> | <code>MPE_FS_ERROR_UNKNOWN_URL</code> |
| <code>MPE_FS_ERROR_UNSUPORT</code> | |

`MPE_FS_ERROR_ALREADY_EXISTS`

`MPE_FS_ERROR_ALREADY_EXISTS` indicates the specified file or directory already exists. `MPE_FS_ERROR_ALREADY_EXISTS` is defined as follows:

```
#define MPE_FS_ERROR_ALREADY_EXISTS (MPE_FS_ERROR_BASE + 2)
```

`MPE_FS_ERROR_DEVICE_FAILURE`

`MPE_FS_ERROR_DEVICE_FAILURE` indicates a lower-level device failed. `MPE_FS_ERROR_DEVICE_FAILURE` is defined as follows:

```
#define MPE_FS_ERROR_DEVICE_FAILURE (MPE_FS_ERROR_BASE + 5)
```

`MPE_FS_ERROR_EOF`

`MPE_FS_ERROR_EOF` indicates the end of the file. `MPE_FS_ERROR_EOF` is defined as follows:

```
#define MPE_FS_ERROR_EOF (MPE_FS_ERROR_BASE + 4)
```

`MPE_FS_ERROR_EVENTS`

`MPE_FS_ERROR_EVENTS` indicates an event failed. `MPE_FS_ERROR_EVENTS` is defined as follows:

```
#define MPE_FS_ERROR_EVENTS MPE_EEVENT
```

`MPE_FS_ERROR_FAILURE`

`MPE_FS_ERROR_FAILURE` indicates a general file-system failure. `MPE_FS_ERROR_FAILURE` is defined as follows:

```
#define MPE_FS_ERROR_FAILURE(MPE_FS_ERROR_BASE + 1)
```

`MPE_FS_ERROR_INVALID_PARAMETER`

`MPE_FS_ERROR_INVALID_PARAMETER` indicates an invalid parameter was specified. `MPE_FS_ERROR_INVALID_PARAMETER` is defined as follows:

```
#define MPE_FS_ERROR_INVALID_PARAMETER MPE EINVAL
```

MPE_FS_ERROR_INVALID_STATE

MPE_FS_ERROR_INVALID_STATE indicates an invalid internal state was passed to a MPEOS function. MPE_FS_ERROR_INVALID_STATE is defined as follows:

```
#define MPE_FS_ERROR_INVALID_STATE (MPE_FS_ERROR_BASE + 6)
```

MPE_FS_ERROR_NOT_FOUND

MPE_FS_ERROR_NOT_FOUND indicates the specified file or directory could not be found. MPE_FS_ERROR_NOT_FOUND is defined as follows:

```
#define MPE_FS_ERROR_NOT_FOUND (MPE_FS_ERROR_BASE + 3)
```

MPE_FS_ERROR_NO_MOUNT

MPE_FS_ERROR_NO_MOUNT indicates there is no device mount available. MPE_FS_ERROR_NO_MOUNT is defined as follows:

```
#define MPE_FS_ERROR_NO_MOUNT (MPE_FS_ERROR_BASE + 8)
```

MPE_FS_ERROR_NOTHING_TO_ABORT

MPE_FS_ERROR_NOTHING_TO_ABORT indicates an abort was specified, but there is no file or directory to abort. MPE_FS_ERROR_NOTHING_TO_ABORT is defined as follows:

```
#define MPE_FS_ERROR_NOTHING_TO_ABORT (MPE_FS_ERROR_BASE + 10)
```

MPE_FS_ERROR_OUT_OF_MEM

MPE_FS_ERROR_OUT_OF_MEM indicates memory could not be allocated. MPE_FS_ERROR_OUT_OF_MEM is defined as follows:

```
#define MPE_FS_ERROR_OUT_OF_MEM MPE_ENOMEM
```

MPE_FS_ERROR_READ_ONLY

MPE_FS_ERROR_READ_ONLY indicates a file or directory is read-only. MPE_FS_ERROR_READ_ONLY is defined as follows:

```
#define MPE_FS_ERROR_READ_ONLY (MPE_FS_ERROR_BASE + 7)
```

MPE_FS_ERROR_SUCCESS

MPE_FS_ERROR_SUCCESS indicates a call was successful. MPE_FS_ERROR_SUCCESS is defined as follows:

```
#define MPE_FS_ERROR_SUCCESS (MPE_SUCCESS)
```

MPE_FS_ERROR_TIMEOUT

MPE_FS_ERROR_TIMEOUT indicates the time-out period for a call has expired. MPE_FS_ERROR_TIMEOUT is defined as follows:

```
#define MPE_FS_ERROR_TIMEOUT MPEETIMEOUT
```

MPE_FS_ERROR_UNKNOWN_URL

MPE_FS_ERROR_UNKNOWN_URL indicates an unknown URL was specified.
MPE_FS_ERROR_UNKNOWN_URL is defined as follows:

```
#define MPE_FS_ERROR_UNKNOWN_URL (MPE_FS_ERROR_BASE + 11)
```

MPE_FS_ERROR_UNSUPPRT

MPE_FS_ERROR_UNSUPPRT indicates a specified function is not currently supported. MPE_FS_ERROR_UNSUPPRT is defined as follows:

```
#define MPE_FS_ERROR_UNSUPPRT (MPE_FS_ERROR_BASE + 9)
```

Data types and structures

The file-system API needs definitions and structures in the following categories:

- ◆ *General*
- ◆ *Broadcast*
- ◆ *Normal*

General

The following general data types and structures are used for both broadcast and standard file system, which are defined in `mpe_file.h`, `mpe_simgr.h`, and `mpe_types.h`, are used by the file-system functions:

-
- ◆ **NOTE:** char type strings are required to be encoded in the Unicode UTF-8 character format.
-

| | |
|-------------------------------|-----------------------------------|
| <code>mpe_Bool</code> | <code>mpe_Dir</code> |
| <code>mpe_File</code> | <code>mpe_FileChangeHandle</code> |
| <code>mpe_FileError</code> | <code>mpe_FileInfo</code> |
| <code>mpe_FileOpenMode</code> | <code>mpe_FileSeekMode</code> |
| <code>mpe_FileStatMode</code> | <code>mpe_SiServiceHandle</code> |
| <code>mpe_Time</code> | |

`mpe_Bool` `mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Dir` `mpe_Dir` identifies an open directory handle. `mpe_Dir` is defined in `mpe_file.h` as follows:

```
typedef void* mpe_Dir;
```

`mpe_File` `mpe_File` identifies an open file handle. `mpe_File` is defined in `mpe_file.h` as follows:

```
typedef void* mpe_File;
```

`mpe_FileChangeHandle`

`mpe_FileChangeHandle` identifies the handle for a file which has changed. `mpe_FileChangeHandle` is defined in `mpe_file.h` as follows:

```
typedef void* mpe_FileChangeHandle;
```

`mpe_FileError` `mpe_FileError` returns error codes from the functions on the MPE file system. `mpe_FileError` is defined in `mpe_file.h` as follows:

```
typedef int32 mpe_FileError;
```

mpe_FileInfo mpe_FileInfo returns file information. mpe_FileInfo is defined in mpe_file.h as follows:

```
struct mpe_FileInfo {
    uint64_t size;
    uint16_t permissions;
    uint16_t type;
    uint8_t priority;
    mpe_Bool isKnown;
    uint32_t sourceID;
    mpe_Time createDate;
    mpe_Time modDate;
    mpe_Time accessDate;
    mpe_Time expDate;
    uint32_t freq;
    uint32_t prog;
    mpe_SiModulationMode qam;
    mpe_Bool hasType;
    uint64_t duration;
    uint32_t numTaps;
    void *buf;
    uint8_t nsap[MPE_FS_NSAP_LENGTH];
    mpe_SiServiceHandle siHandle;
    int32 numReadAccess0rgs;
    int16 *readAccess0rgs;
    int32 numWriteAccess0rgs;
    uint16_t *writeAccess0rgs;
};
```

where

size indicates the size of the file in bytes.

permissions specifies the access permissions for the specified file. More than one permission can be assigned to this field. Currently, the following values are supported for permissions:

MPE_FS_PERM_GROUP_READ

 indicates object has group read permissions.

MPE_FS_PERM_OWNER_READ

 indicates object has owner read permissions.

MPE_FS_PERM_OWNER_WRITE

 indicates object has owner write permissions.

MPE_FS_PERM_WORLD_READ

 indicates object has all read permissions.

MPE_FS_PERM_WORLD_WRITE

 indicates object has all write permissions.

| | |
|-------------------|---|
| type | indicates the type of file. Currently, the following values are supported for type: |
| | MPE_FS_TYPE_DIR indicates the object type is a directory. |
| | MPE_FS_TYPE_FILE indicates the object type is a file. |
| | MPE_FS_TYPE_OTHER indicates the object type is known, but is not a file or directory (for example, stream). |
| | MPE_FS_TYPE_STREAM indicates the object type is a stream. |
| | MPE_FS_TYPE_STREAMEVENT indicates the object type is a stream event. |
| | MPE_FS_TYPE_UNKNOWN indicates the object type is unknown. |
| priority | indicates the priority level. Currently, the following values are supported for priority: |
| | MPE_FS_PRIOR_HI indicates a high-priority file. |
| | MPE_FS_PRIOR_LOW indicates a low-priority file. |
| | MPE_FS_PRIOR_MED indicates a medium-priority file. |
| isKnown | determines if the parent directory is loaded. If isKnown is TRUE, it indicates the parent directory is loaded. If isKnown is FALSE, it indicates the parent directory is not loaded. mpe_Bool is described in the <i>mpe_Bool</i> section. |
| sourceID | indicates the source ID this file is coming on (BFS only). |
| createDate | indicates the day the file was created. mpe_Time is described in the <i>mpe_Time</i> section. |
| modDate | indicates the last date the file was modified. mpe_Time is described in the <i>mpe_Time</i> section. |
| accessDate | indicates the last date the file was accessed. Currently, this is not being used. mpe_Time is described in the <i>mpe_Time</i> section. |
| expDate | indicates the expiration date for the file. mpe_Time is described in the <i>mpe_Time</i> section. |
| freq | indicates the frequency for GET_TUNINGINFO . |
| prog | indicates the program number for GET_TUNINGINFO . |
| qam | indicates the qam mode for GET_TUNINGINFO . SiModulationMode is described in the <i>mpe_SiModulationMode</i> section. |

| | |
|---------------------------------------|--|
| <code>hasType</code> | determines if the type requested matches the type. If <code>hasType</code> is TRUE, it indicates the type matched. If <code>hasType</code> is FALSE, it indicates the type did not matched. <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |
| <code>duration</code> | indicates the duration of the stream in milliseconds. |
| <code>numTaps</code> | indicates the number of taps in the stream. |
| <code>buf</code> | is a pointer to the caller's buffer (<code>mpe_FileInfo.size</code>). |
| <code>nsap[MPE_FS_NSAP_LENGTH]</code> | indicates the Network Service Access Point (NSAP) address length in bytes. |
| <code>siHandle</code> | indicates the new SI service handle to set. <code>mpe_SiServiceHandle</code> is described in the <i>mpe_SiServiceHandle</i> section. |
| <code>numReadAccess0rgs</code> | specifies the number of organization IDs allowed with read access. OCAP applications have an assigned organization ID based on the organization that wrote and delivered the application. File access may be limited to certain organizations. |
| <code>readAccess0rgs</code> | is a pointer to a dynamically allocated array of organization IDs with read access. |
| <code>numWriteAccess0rgs</code> | specifies the number of organization IDs allowed with read access. |
| <code>writeAccess0rgs</code> | is a pointer to a dynamically allocated array of organization IDs with write access. |
| <code>mpe_FileOpenMode</code> | <code>mpe_FileOpenMode</code> is used with the file open bit-field parameters. <code>mpe_FileOpenMode</code> is defined in <code>mpe_file.h</code> as follows: |
| | <code>typedef uint32_t mpe_FileOpenMode;</code> |
| <code>mpe_FileSeekMode</code> | <code>mpe_FileSeekMode</code> searches for a specific position in the file and should be used with the file seek mode parameters. <code>mpe_FileSeekMode</code> is defined in <code>mpe_file.h</code> as follows: |
| | <code>typedef uint32_t mpe_FileSeekMode;</code> |
| <code>mpe_FileStatMode</code> | <code>mpe_FileStatMode</code> returns and sets the status calls. <code>mpe_FileStatMode</code> is defined in <code>mpe_file.h</code> as follows: |
| | <code>typedef uint32_t mpe_FileStatMode;</code> |

mpe_SiServiceHandle

`mpe_SiServiceHandle` returns and sets an SI service handle.
`mpe_SiServiceHandle` is defined in `mpe_simg.h` as follows:

```
typedef uint32_t mpe_SiServiceHandle;
```

mpe_Time

`mpe_Time` specifies the time in seconds. `mpe_Time` is defined in `mpe_time.h` as follows:

```
typedef os_Time mpe_Time;
```

Broadcast

The following broadcast data types and structures, which are defined in `mpe_file.h`, `mpe_types.h`, `mpeos_ed.h`, `mpeos_event.h`, `mpeos_file.h`, `mpeos_file_romfs.h`, `mpeos_si.h`, and `mpeos_time.h`, are used by the file-system functions:

- ◆ **NOTE:** char type strings are required to be encoded in the Unicode UTF-8 character format.

| | |
|-----------------------------------|------------------------------|
| <code>mpe_DirInfo</code> | <code>mpe_DirStatMode</code> |
| <code>mpe_DirUrl</code> | <code>mpe_EventQueue</code> |
| <code>mpe_SiModulationMode</code> | <code>mpe_Stream</code> |
| <code>mpe_StreamEventInfo</code> | |

`mpe_DirInfo`

`mpe_DirInfo` indicates whether the connection is available or the path to the directory. `mpe_DirInfo` is defined in `mpe_file.h` as follows:

```
typedef union {
    mpe_Bool isConnectAvail;
    char path[MPE_FS_MAX_PATH+1];
} mpe_DirInfo;
```

where

`isConnectAvail`

specifies whether the connection is available. `mpe_Bool` is defined in the `mpe_Bool` section.

`path`

specifies the file-system path.

`mpe_DirStatMode`

`mpe_DirStatMode` specifies the status of the directory. `mpe_DirStatMode` is defined in `mpe_file.h` as follows:

```
typedef uint32_t mpe_DirStatMode;
```

`mpe_DirUrl`

`mpe_DirUrl` identifies the location and the ID of the directory. `mpe_DirUrl` is defined in `mpe_file.h` as follows:

```
typedef struct mpe_DirUrl {
    const char *url;
    mpe_SiServiceHandle siHandle;
    uint32_t carouselId;
} mpe_DirUrl;
```

where

`url` is a pointer to the Universal Resource Location (URL), not the file-system path, to the specified object.

`siHandle` indicates the SI service handle. `mpe_SiServiceHandle` is described in the `mpe_SiServiceHandle` section.

`carouselId` indicates the carousel ID for the specified object.

mpe_EventQueue mpe_EventQueue defines the event queue type. mpe_EventQueue is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

mpe_SiModulationMode

mpe_SiModulationMode defines the modulation mode used by the head-end for tuning parameters. mpe_SiModulationMode is defined in `mpeos_si.h` as follows:

```
typedef enum mpe_SiModulationMode {
    MPE_SI_MODULATION_UNKNOWN=0,
    MPE_SI_MODULATION_QPSK,
    MPE_SI_MODULATION_BPSK,
    MPE_SI_MODULATION_OQPSK,
    MPE_SI_MODULATION_VSB8,
    MPE_SI_MODULATION_VSB16,
    MPE_SI_MODULATION_QAM16,
    MPE_SI_MODULATION_QAM32,
    MPE_SI_MODULATION_QAM64,
    MPE_SI_MODULATION_QAM80,
    MPE_SI_MODULATION_QAM96,
    MPE_SI_MODULATION_QAM112,
    MPE_SI_MODULATION_QAM128,
    MPE_SI_MODULATION_QAM160,
    MPE_SI_MODULATION_QAM192,
    MPE_SI_MODULATION_QAM224,
    MPE_SI_MODULATION_QAM256,
    MPE_SI_MODULATION_QAM320,
    MPE_SI_MODULATION_QAM384,
    MPE_SI_MODULATION_QAM448,
    MPE_SI_MODULATION_QAM512,
    MPE_SI_MODULATION_QAM640,
    MPE_SI_MODULATION_QAM768,
    MPE_SI_MODULATION_QAM896,
    MPE_SI_MODULATION_QAM1024,
    MPE_SI_MODULATION_QAM_NTSC = 255
} mpe_SiModulationMode;
```

where

MPE_SI_MODULATION_UNKNOWN

specifies an unknown modulation mode.

MPE_SI_MODULATION_QPSK

specifies Quadrature Phase-Shift Keying (QPSK) modulation mode.

MPE_SI_MODULATION_BPSK

specifies Binary Phase-Shift Keying (BPSK) modulation mode.

MPE_SI_MODULATION_OQPSK
specifies Offset Quadrature Phase-Shift Keying (OQPSK) modulation mode.

MPE_SI_MODULATION_VSB8
specifies 8-level Vestigial Sideband (VSB8) modulation mode.

MPE_SI_MODULATION_VSB16
specifies 16-level Vestigial Sideband (VSB16) modulation mode.

MPE_SI_MODULATION_QAM16
specifies Quadrature Amplitude Modulation (QAM) QAM16 modulation mode.

MPE_SI_MODULATION_QAM32
specifies QAM32 modulation mode.

MPE_SI_MODULATION_QAM64
specifies QAM64 modulation mode.

MPE_SI_MODULATION_QAM80
specifies QAM80 modulation mode.

MPE_SI_MODULATION_QAM96
specifies QAM96 modulation mode.

MPE_SI_MODULATION_QAM112
specifies QAM112 modulation mode.

MPE_SI_MODULATION_QAM128
specifies QAM128 modulation mode.

MPE_SI_MODULATION_QAM160
specifies QAM160 modulation mode.

MPE_SI_MODULATION_QAM192
specifies QAM192 modulation mode.

MPE_SI_MODULATION_QAM224
specifies QAM224 modulation mode.

MPE_SI_MODULATION_QAM256
specifies QAM256 modulation mode.

MPE_SI_MODULATION_QAM320
specifies QAM320 modulation mode.

MPE_SI_MODULATION_QAM384
specifies QAM384 modulation mode.

MPE_SI_MODULATION_QAM448
specifies QAM448 modulation mode.

MPE_SI_MODULATION_QAM512
specifies QAM512 modulation mode.

MPE_SI_MODULATION_QAM640
specifies QAM640 modulation mode.

MPE_SI_MODULATION_QAM768
specifies QAM768 modulation mode.

MPE_SI_MODULATION_QAM896
specifies QAM896 modulation mode.

MPE_SI_MODULATION_QAM1024
specifies QAM1024 modulation mode.

MPE_SI_MODULATION_QAM_NTSC
specifies National Television System(s) Committee (NTSC) (or analog) modulation mode and is assigned 255.

mpe_Stream mpe_Stream specifies the pointer to a stream. mpe_Stream is defined in mpe_file.h as follows:

```
typedef void *mpe_Stream;
```

mpe_StreamEventInfo

mpe_StreamEventInfo specifies the pointer to a stream.
mpe_StreamEventInfo is defined in mpe_file.h as follows:

```
typedef struct {  
    uint32_t eventID;  
    char name[MPE_FS_MAX_PATH + 1];  
} mpe_StreamEventInfo;
```

where

eventID identifies the specific event.

name identifies the name of the event.

Normal

The following normal data type and structure, which are defined in `mpe_file.h`, are used by the file-system functions:

- ◆ **NOTE:** `char` type strings are required to be encoded in the Unicode UTF-8 character format.

| | |
|---------------------------|-------------------------------------|
| <code>mpe_DirEntry</code> | <code>mpeos_filesys_ftable_t</code> |
|---------------------------|-------------------------------------|

`mpe_DirEntry`

`mpe_DirEntry` specifies the path to the directory. `mpe_DirEntry` is defined in `mpe_file.h` as follows:

```
typedef struct mpe_DirEntry {
    char name[MPE_FS_MAX_PATH+1];
    uint64_t fileSize;
    mpe_Bool isDir;
} mpe_DirEntry;
```

where

`name[MPE_FS_MAX_PATH+1]`

specifies the path to the file or directory.

`fileSize` if the directory entry is a file, it specifies the size of the file in bytes.

`isDir` is a flag indicating the directory entry is a subdirectory or a file. `mpe_Bool` is defined in the *mpe_Bool* section.

`mpeos_filesys_ftable_t`

`mpeos_filesys_ftable_t` is the function table for individual drivers. The file system module calls into these functions when performing operations. Each file-system driver (which could be multiple drivers) must implement a set of functions, each with a specific parameter list and functionality as appropriate for that file system. After the functions are implemented, a `mpeos_filesys_ftable_t` is created referencing those file system functions. Unlike other modules that have specific functions to implement, the file system has functions to implement for each driver.

`mpeos_filesys_ftable_t` is defined in `mpeos_file.h` as follows:

```
typedef struct mpeos_filesys_ftable_t {
    void (*mpeos_init_ptr)(const char* mountPoint);
    mpe_FileError (*mpeos_fileOpen_ptr)(const char* fileName,
                                         mpe_FileOpenMode openMode, mpe_File* returnHandle);
    mpe_FileError (*mpeos_fileClose_ptr)(mpe_File handle);
    mpe_FileError (*mpeos_fileRead_ptr)(mpe_File handle,
                                         uint32_t* count, void* buffer);
    mpe_FileError (*mpeos_fileWrite_ptr)(mpe_File handle,
                                         uint32_t* count, void* buffer);
    mpe_FileError (*mpeos_fileSeek_ptr)(mpe_File handle,
                                         mpe_FileSeekMode seekMode, int64* offset);
    mpe_FileError (*mpeos_fileSync_ptr)(mpe_File handle);
    mpe_FileError (*mpeos_fileGetStat_ptr)
        (const char* fileName, mpe_FileStatMode mode,
         mpe_FileInfo *info);
```

```

mpe_FileError (*mpeos_fileSetStat_ptr)
    (const char* fileName, mpe_FileStatMode mode,
     mpe_FileInfo *info);
mpe_FileError (*mpeos_fileGetFStat_ptr)
    (mpe_File handle, mpe_FileStatMode mode,
     mpe_FileInfo *info);
mpe_FileError (*mpeos_fileSetFStat_ptr)
    (mpe_File handle, mpe_FileStatMode mode,
     mpe_FileInfo *info);
mpe_FileError (*mpeos_fileDelete_ptr)
    (const char* fileName);
mpe_FileError (*mpeos_fileRename_ptr)(const char* oldName,
                                       const char* newName);
mpe_FileError (*mpeos_dirOpen_ptr)
    (const char* name, mpe_Dir* returnHandle);
mpe_FileError (*mpeos_dirRead_ptr)
    (mpe_Dir handle, mpe_DirEntry* dirEnt);
mpe_FileError (*mpeos_dirClose_ptr)(mpe_Dir handle);
mpe_FileError (*mpeos_dirDelete_ptr)(const char* dirName);
mpe_FileError (*mpeos_dirRename_ptr)(const char* oldName,
                                      const char* newName);
mpe_FileError (*mpeos_dirCreate_ptr)(const char* dirName);
mpe_FileError (*mpeos_dirMount_ptr)
    (const mpe_DirUrl *dirUrl);
mpe_FileError (*mpeos_dirUnmount_ptr)
    (const mpe_DirUrl *dirUrl);
mpe_FileError (*mpeos_dirGetUStat_ptr)
    (const mpe_DirUrl *dirUrl, mpe_DirStatMode mode,
     mpe_DirInfo *info);
mpe_FileError (*mpeos_dirSetUStat_ptr)
    (const mpe_DirUrl *dirUrl, mpe_DirStatMode mode,
     mpe_DirInfo *info);
mpe_FileError (*mpeos_streamOpen_ptr)
    (const char *fileName, mpe_Stream *streamHandlePtr);
mpe_FileError (*mpeos_streamClose_ptr)
    (mpe_Stream streamHandle);
mpe_FileError (*mpeos_streamReadEvent_ptr)
    (mpe_Stream streamHandle, mpe_StreamEventInfo *event);
mpe_FileError (*mpeos_streamGetNumTaps_ptr)
    (mpe_Stream streamHandle, uint16_t tapType,
     uint32_t *numTaps);
mpe_FileError (*mpeos_streamReadTap_ptr)
    (mpe_Stream streamHandle, uint16_t tapType,
     uint32_t tapNumber, uint16_t *tap, uint16_t *tapID);
mpe_FileError (*mpeos_filePrefetch_ptr)
    (const char *fileName);
mpe_FileError (*mpeos_filePrefetchModule_ptr)
    (const char *mountpoint, const char *moduleName);
mpe_FileError (*mpeos_fileAddDII_ptr)
    (const char *mountpoint, uint16_t diiId, uint16_t
     assocTag);

```

```
mpe_FileError (*mpeos_fileSetChangeListener_ptr)
    (const char *fileName, mpe_EventQueue , void *
     act, mpe_FileChangeHandle *handle);
mpe_FileError (*mpeos_fileRemoveChangeListener_ptr)
    (mpe_FileChangeHandle handle);
} mpeos_filesys_ftable_t;
```

General file system functions

The following functions are implemented once per port:

`mpeos_filesysGetDefaultDir`

specifies the size and location of the default directory.

`mpeos_filesysInit`

initializes the port-specific file system.

`mpeos_filesysGetDefaultDir`

specifies the size and location of the default directory.

syntax mpe_FileError mpeos_filesysGetDefaultDir(
 char * *buf*,
 uint32_t *bufSz*);

| | | |
|---------------------|--------------|--|
| parameter(s) | <i>buf</i> | is an input pointer to the caller-allocated string buffer where the default system directory will be copied. |
| | <i>bufSz</i> | specifies the size in bytes of the buffer. |

value returned If the call is successful, `mpeos_filesysGetDefaultDir()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

MPE FS ERROR INVALID PARAMETER

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NOT_FOUND

indicates the file path could not be found.

MPE_FS_ERROR_OUT_OF_MEM

indicates there is not enough memory for the default path string.

description mpeos_filesysGetDefaultDir should get the size and location of the default system directory.

related function(s) none

mpeos_filesysInit

initializes the port-specific file system.

syntax void mpeos_filesysInit(
 void (*mpe_filesysRegisterDriver)(
 const mpeos_filesys_ftable_t * *ftable*,
 const char * *mountPoint*));

parameter(s) *mpe_filesysRegisterDriver*
 is an input pointer to the file system register driver.

ftable is an input pointer to the function table.
 mpeos_filesys_ftable_t is described in the
 mpeos_filesys_ftable_t section.

mountPoint is an input pointer to the operating-system specific mount.

value returned none

description mpeos_filesys_init should be responsible for registering all port-specific, file-system implementations. Use the *mpe_filesysRegister()* driver function to register a MPEOS file-system function table to a particular mount point.

related function(s) none

File system-specific functions

Each port may need to support and implement many file systems. For example, a single hardware platform may support a Network File System (NFS), a Broadcast File System (BFS), and a local hard disk or RAM disk-based file system.

-
- ◆ **PORTING NOTE:** The OCAP stack contains an object carousel implementation and there are several functions which exist only to support the object carousel. Porters are not expected to implement these functions, they are documented only for completeness. The object carousel functions are:

| | |
|---|------------------------------------|
| <code>mpeos_fileAddDII()</code> | <code>mpeos_filePrefetch()</code> |
| <code>mpeos_filePrefetchModule()</code> | <code>mpeos_steamClose()</code> |
| <code>mpeos_streamGetNumTaps()</code> | <code>mpeos_streamOpen()</code> |
| <code>mpeos_steamReadEvent()</code> | <code>mpeos_streamReadTap()</code> |

A separate function table for each unique file system type needs to be implemented for the following functions:

`mpeos_dirClose`
destroys a handle to an open directory.

`mpeos_dirCreate`
creates a new directory.

`mpeos_dirDelete`
deletes an existing directory.

`mpeos_dirGetUStat`
gets information about a directory based on the URL (BFS only).

`mpeos_dirMount`
mounts a file-system based on its URL (BFS only).

`mpeos_dirOpen`
creates a new handle to the directory.

`mpeos_dirRead`
reads the next directory entry from an opened directory.

`mpeos_dirRename`
renames a directory.

`mpeos_dirSetUStat`
sets information about a directory based on its URL (BFS only).

`mpeos_dirUnmount`
removes the mount of a file-system based on its URL (BFS only).

`mpeos_fileClose`
destroys the handle to an open file.

`mpeos_fileDelete`
deletes the specified file.

`mpeos_fileGetFStat`
gets information about a file based on its open handle.

`mpeos_fileGetStat`
gets information about a file based on the pathname.

`mpeos_fileOpen`
creates a new handle to a file.

`mpeos_fileRead`
reads data from an opened file.

`mpeos_fileRename`
renames a file.

`mpeos_fileRemoveChangeListener`
removes an event listener for the specified file.

`mpeos_fileSetChangeListener`
registers a listener.

`mpeos_fileSetFStat`
sets information about a file based on its open handle.

`mpeos_fileSetStat`
sets information about a file based on its pathname.

`mpeos_fileSync`
synchronizes file contents on the storage device with any pending file data.

`mpeos_fileWrite`
writes data to an open file.

`mpeos_init` initializes this MPEOS operating-system specific file-system interface.

mpeos_dirClose

destroys a handle to an open directory.

syntax mpe_FileError mpeos_dirClose(mpe_Dir dirHandle);

parameter(s) *dirHandle* specifies the handle to directory to close. *dirHandle* is returned by a previous call to mpeos_dirOpen(). *mpe_Dir* is described in the *mpe_Dir* section.

value returned If the call is successful, mpeos_dirClose() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE
indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER
indicates an invalid parameter (for example, out-of-range values, null pointers, or unknown values).

MPE_FS_ERROR_NOT_FOUND
indicates the directory path could not be found.

description mpeos_dirClose() should close the previously opened directory specified by *dirHandle*.

related function(s) mpeos_dirOpen

mpeos_dirCreate

creates a new directory.

syntax mpe_FileError mpeos_dirCreate(const char * *dirName*);

parameter(s) *dirName* is an input pointer to the path of a directory to create.

value returned If the call is successful, mpeos_dirCreate() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE

indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NO_MOUNT

indicates no device mount was found to handle the path.

MPE_FS_ERROR_NOT_FOUND

indicates the directory path could not be found.

MPE_FS_ERROR_OUT_OF_MEM

indicates memory could not be allocated from the system.

description mpeos_dirCreate() should create a new directory.

related function(s) mpeos_dirDelete
mpeos_dirRename

mpeos_dirDelete

deletes an existing directory.

syntax mpe_FileError mpeos_dirDelete(const char * *dirName*);

parameter(s) *dirName* is an input pointer to the path of an existing directory to delete. *dirName* is returned by a previous call to mpeos_dirOpen().

value returned If the call is successful, mpeos_dirDelete() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE

indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NO_MOUNT

indicates no device mount was found to handle the path.

MPE_FS_ERROR_NOT_FOUND

indicates the directory path could not be found.

MPE_FS_ERROR_OUT_OF_MEM

indicates memory could not be allocated from the system.

description mpeos_dirDelete() should delete the directory specified by *dirName*.

related function(s) mpeos_dirCreate
mpeos_dirOpen
mpeos_dirRename

mpeos_dirGetUStat

gets information about a directory based on the URL (BFS only).

syntax mpe_FileError mpeos_dirGetUStat(
 const mpe_DirUrl * *dirUrl*,
 mpe_DirStatMode *mode*,
 mpe_DirInfo * *info*);

| | | |
|---------------------|-----------------------------|---|
| parameter(s) | <i>dirUrl</i> | is an input pointer to the URL of an existing directory in which to return the status information. The OCAP stack automatically fills in <i>dirUrl</i> . <i>mpe_DirUrl</i> is described in the <i>mpe_DirUrl</i> section. |
| | <i>mode</i> | specifies the information being requested. <i>mpe_DirStatMode</i> is described in the <i>mpe_DirStatMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | MPE_FS_STAT_CONNECTIONAVAIL | gets a TRUE or FALSE value based on the connection status for the specified network directory. |
| | MPE_FS_STAT_MOUNTPATH | gets the absolute system mount paths for the indicated object path. |
| | <i>info</i> | is an output pointer to the buffer in which to return the requested information. <i>mpe_DirInfo</i> is described in the <i>mpe_DirInfo</i> section. |

value returned If the call is successful, *mpeos_dirGetUStat()* should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

| | |
|--------------------------------|---|
| MPE_FS_ERROR_DEVICE_FAILURE | indicates the underlying call failed. |
| MPE_FS_ERROR_INVALID_PARAMETER | indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.). |
| MPE_FS_ERROR_UNKNOWNURL | indicates the URL is not supported. |
| MPE_FS_ERROR_UNSUPPORT | indicates the requested information <i>mode</i> is not currently supported for the file. |

description *mpeos_dirGetUStat()* should get information about a directory based on the URL.

related function(s) *mpeos_dirSetUStat*

mpeos_dirMount

mounts a file-system based on its URL (BFS only).

syntax mpe_FileError mpeos_dirMount(const mpe_DirUrl * *dirUrl*);

parameters *dirUrl* is an input pointer to the URL of an existing directory to mount. The OCAP stack automatically fills in *dirUrl*. *mpe_DirUrl* is described in the *mpe_DirUrl* section.

value returned If the call is successful, *mpeos_dirMount()* should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE
indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_UNKNOWNURL
indicates the URL is not supported.

MPE_FS_ERROR_NOT_FOUND
indicates the directory path could not be found.

description *mpeos_dirMount()* should create a mount point for the directory URL.

related function(s) *mpeos_dirGetUStat*
mpeos_dirUnmount

mpeos_dirOpen

creates a new handle to the directory.

syntax mpe_FileError mpeos_dirOpen(
 const char * *name*,
 mpe_Dir * *returnHandle*);

parameter(s) *name* is an input pointer to the path of an existing directory to open.

returnHandle is an output pointer to the handle of the opened directory.
mpe_Dir is described in the *mpe_Dir* section.

value returned If the call is successful, *mpeos_dirOpen()* should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE
 indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER
 indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NO_MOUNT
 indicates no device mount was found to handle the path.

MPE_FS_ERROR_NOT_FOUND
 indicates the directory path could not be found.

MPE_FS_ERROR_OUT_OF_MEM
 indicates memory could not be allocated from the system.

description *mpeos_dirOpen()* should create an access handle to a directory based on the *path*.

related function(s) *mpeos_dirClose*

mpeos_dirRead

reads the next directory entry from an opened directory.

syntax mpe_FileError mpeos_dirRead(
 mpe_Dir *dirHandle*,
 mpe_DirEntry * *entry*);

| | | |
|---------------------|------------------|---|
| parameter(s) | <i>dirHandle</i> | specifies the handle of the previously opened directory. <i>dirHandle</i> is returned by a previous call to <code>mpeos_dirOpen()</code> . <i>mpe_Dir</i> is described in the <i>mpe_Dir</i> section. |
| | <i>entry</i> | is an output pointer to a directory entry structure in which to read the directory entry data. <i>mpe_DirEntry</i> is described in the <i>mpe_DirEntry</i> section. |

value returned If the call is successful, `mpeos_dirRead()` should return `MPE_FS_ERROR_SUCCESS` and return a pointer to directory entry data in *entry*. Otherwise, it should return one of the following error codes:

| | |
|--------------------------------|---|
| MPE_FS_ERROR_DEVICE_FAILURE | indicates the underlying call failed. |
| MPE_FS_ERROR_INVALID_PARAMETER | indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.). |
| MPE_FS_ERROR_NOT_FOUND | indicates the directory path could not be found or all the objects in the directory have been read. |
| MPE_FS_ERROR_OUT_OF_MEM | indicates memory could not be allocated from the system. |

description `mpeos_dirRead()` should read the directory entry data from a previously opened directory. The first call gets the first object's information, the second call gets the second object's information, etc.

related function(s) `mpeos_dirOpen`

mpeos_dirRename

renames a directory.

syntax `mpe_FileError mpeos_dirRename(
 const char * oldDirName,
 const char * newDirName);`

parameter(s) `oldDirName` is an input pointer to the path of an existing directory to rename and/or move.

`newDirName` is an input pointer to the new path and/or name.

value returned If the call is successful, `mpeos_dirRename()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`

indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_FS_ERROR_NO_MOUNT`

indicates no device mount was found to handle the path.

`MPE_FS_ERROR_NOT_FOUND`

indicates the directory path could not be found.

`MPE_FS_ERROR_OUT_OF_MEM`

indicates memory could not be allocated from the system.

description `mpeos_dirRename()` should rename and/or move the directory.

related function(s) `mpeos_dirCreate`
`mpeos_dirDelete`

mpeos_dirSetUStat

sets information about a directory based on its URL (BFS only).

syntax `mpe_FileError mpeos_dirSetUStat(const mpe_DirUrl * dirUrl, mpe_DirStatMode mode, mpe_DirInfo * info);`

| | | |
|---------------------|--|---|
| parameter(s) | <i>dirUrl</i> | is an input pointer to the URL of an existing directory in which to update the status information. The OCAP stack automatically fills in <i>dirUrl</i> . <i>mpe_DirUrl</i> is described in the <i>mpe_DirUrl</i> section. |
| | <i>mode</i> | specifies the information being updated. <i>mpe_DirStatMode</i> is described in the <i>mpe_DirStatMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | <code>MPE_FS_STAT_CONNECTIONAVAIL</code> | gets a TRUE or FALSE value based on the connection status for the specified network directory. |
| | <code>MPE_FS_STAT_MOUNTPATH</code> | gets the absolute system mount paths for the indicated object path. |
| | <i>info</i> | is an input pointer to the buffer in which to return the updated information. <i>mpe_DirInfo</i> is described in the <i>mpe_DirInfo</i> section. |

value returned If the call is successful, `mpeos_dirSetUStat()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

| | |
|---|---|
| <code>MPE_FS_ERROR_DEVICE_FAILURE</code> | indicates the underlying call failed. |
| <code>MPE_FS_ERROR_INVALID_PARAMETER</code> | indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.). |
| <code>MPE_FS_ERROR_NOT_FOUND</code> | indicates the directory path could not be found. |
| <code>MPE_FS_ERROR_UNKNOWNURL</code> | indicates the URL is not supported. |
| <code>MPE_FS_ERROR_UNSUPPRT</code> | indicates the requested information <i>mode</i> is not currently supported for the file. |

description `mpeos_dirSetUStat()` should update specified information about a directory URL.

related function(s) `mpeos_dirGetUStat`

mpeos_dirUnmount

removes the mount of a file-system based on its URL (BFS only).

syntax mpe_FileError mpeos_dirUnmount(const mpe_DirUrl * *dirUrl*);

parameters *dirUrl* is an input pointer to the URL of an existing directory to unmount. *dirUrl* is returned from a previous call to mpeos_dirMount(). The OCAP stack automatically fills in *dirUrl*. *mpe_DirUrl* is described in the *mpe_DirUrl* section.

value returned If the call is successful, mpeos_dirUnmount() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE
indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NOT_FOUND
indicates the directory path could not be found.

MPE_FS_ERROR_UNKNOWNURL
indicates the URL is not supported.

description mpeos_dirUnmount() should unmount the previously mounted URL specified by *dirUrl*.

related function(s) mpeos_dirMount

mpeos_fileClose

destroys the handle to an open file.

syntax mpe_FileError mpeos_fileClose(mpe_File fileHandle);

parameter(s) *fileHandle* specifies an open file access handle. *fileHandle* is returned by a previous call to mpeos_fileOpen(). *mpe_File* is described in the *mpe_File* section.

value returned If the call is successful, mpeos_fileClose() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE
indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NOT_FOUND
indicates the file path could not be found.

description mpeos_fileClose() should destroy an opened access handle to a file.

related function(s) mpeos_fileOpen

mpeos_fileDelete

deletes the specified file.

syntax `mpe_FileError mpeos_fileDelete(const char * fileName);`

parameter(s) `fileName` is an input pointer to the path of an existing file to delete.

value returned If the call is successful, `mpeos_fileDelete()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`

indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_FS_ERROR_NO_MOUNT`

indicates no device mount was found to handle the path.

`MPE_FS_ERROR_NOT_FOUND`

indicates the file path could not be found.

description `mpeos_fileDelete()` should delete the file specified by `fileName`.

related function(s) `mpeos_fileRename`

mpeos_fileGetFStat

gets information about a file based on its open handle.

syntax mpe_FileError mpeos_fileGetFStat(
 mpe_File *fileHandle*,
 mpe_FileStatMode *mode*,
 mpe_FileInfo * *info*);

| | | |
|---------------------|------------------------|--|
| parameter(s) | <i>fileHandle</i> | specifies an open file access handle. <i>fileHandle</i> is returned by a previous call to <code>mpeos_fileOpen()</code> . <code>mpe_File</code> is described in the <i>mpe_File</i> section. |
| | <i>mode</i> | specifies the information being requested. <code>mpe_FileStatMode</code> is described in the <i>mpe_FileStatMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | MPE_FS_STAT_CREATEDATE | gets the date the file was created. |
| | MPE_FS_STAT_EXPDATE | gets or sets the expiration date. |
| | MPE_FS_STAT_ISKNOWN | specifies the value for the object is known. |
| | MPE_FS_STAT_MODDATE | gets the last date the file was modified. |
| | MPE_FS_STAT_ORG_ACCESS | gets the list of organization IDs that have read and/or write access. |
| | MPE_FS_STAT_PERM | gets the permission for a Unix style file or directory. |
| | MPE_FS_STAT_PRIOR | gets the priority of the file. |
| | MPE_FS_STAT_SIZE | gets the size of the file. |
| | MPE_FS_STAT_SOURCEID | gets the source identification of the file location. |
| | MPE_FS_STAT_TYPE | gets the type of file. |
| <i>info</i> | | is an output pointer to the buffer in which to return the requested information. <code>mpe_FileInfo</code> is described in the <i>mpe_FileInfo</i> section. |

value returned

If the call is successful, `mpeos_fileGetFStat()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`

indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_FS_ERROR_NOT_FOUND`

indicates the file path could not be found.

`MPE_FS_ERROR_UNSUPPOT`

indicates the requested information *mode* is not currently supported for the file.

description

`mpeos_fileGetFStat()` should allow the caller to retrieve a specific piece of information (stat) about a previously opened file.

related function(s)

`mpeos_fileGetStat`

`mpeos_fileOpen`

`mpeos_fileSetFStat`

`mpeos_fileSetStat`

mpeos_fileGetStat

gets information about a file based on the pathname.

syntax mpe_FileError mpeos_fileGetStat(
 const char * *fileName*,
 mpe_FileStatMode *mode*,
 mpe_FileInfo * *info*);

| | | |
|---------------------|------------------------|---|
| parameter(s) | <i>fileName</i> | is an input pointer to the path of an existing file from which to get information. |
| | <i>mode</i> | specifies the information being requested. mpe_FileStatMode is described in the <i>mpe_FileStatMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | MPE_FS_STAT_CREATEDATE | gets the date the file was created. |
| | MPE_FS_STAT_EXPDATE | gets the expiration date. |
| | MPE_FS_STAT_ISKNOWN | specifies the value for the object is known. |
| | MPE_FS_STAT_MODDATE | gets the last date the file was modified. |
| | MPE_FS_STAT_ORG_ACCESS | gets the list of organization IDs that have read and/or write access. |
| | MPE_FS_STAT_PERM | gets the permission for a Unix style file. |
| | MPE_FS_STAT_PRIOR | gets the priority of the file. |
| | MPE_FS_STAT_SIZE | gets the size of the file. |
| | MPE_FS_STAT_SOURCEID | gets the source identification of the file location. |
| | MPE_FS_STAT_TYPE | gets the type of file. |
| <i>info</i> | | is an output pointer to the buffer in which to return the requested information. mpe_FileInfo is described in the <i>mpe_FileInfo</i> section. |

value returned If the call is successful, `mpeos_fileGetStat()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

- `MPE_FS_ERROR_DEVICE_FAILURE`
indicates the underlying call failed.
- `MPE_FS_ERROR_INVALID_PARAMETER`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).
- `MPE_FS_ERROR_NO_MOUNT`
indicates no device mount was found to handle the path.
- `MPE_FS_ERROR_NOT_FOUND`
indicates the file path could not be found.
- `MPE_FS_ERROR_OUT_OF_MEM`
indicates memory could not be allocated from the system.
- `MPE_FS_ERROR_UNSUPPORTED`
indicates the requested information *mode* is not currently supported for the file.

description `mpeos_fileGetStat()` should get information about the file specified by *fileName*.

related function(s) `mpeos_fileGetFStat`
`mpeos_fileOpen`
`mpeos_fileSetFStat`
`mpeos_fileSetStat`

mpeos_fileOpen

creates a new handle to a file.

syntax mpe_FileError mpeos_fileOpen(
 const char * *fileName*,
 mpe_FileOpenMode *openMode*,
 mpe_File * *returnHandle*);

parameter(s) *fileName* is an input pointer to the path of an existing file to open.

openMode specifies the type of access requested for opening the file.
mpe_FileOpenMode is described in the *mpe_FileOpenMode* section. Currently, the following values are supported for *mode*:

MPE_FS_OPEN_APPEND

opens the file so that writing begins at the end of the file. MPE_FS_OPEN_APPEND is used only in conjunction with MPE_FS_OPEN_WRITE or MPE_FS_OPEN_READWRITE.

MPE_FS_OPEN_CAN_CREATE

creates a file if necessary.

MPE_FS_OPEN_MUST_CREATE

creates a file.

MPE_FS_OPEN_READ

opens a file for read-only.

MPE_FS_OPEN_READWRITE

opens a file for reading and writing.

MPE_FS_OPEN_TRUNCATE

opens the file and deletes any existing content and writes to the beginning of the file.

MPE_FS_OPEN_WRITE

opens a file for writing.

returnHandle is an output pointer to a buffer in which the file handle is returned. *mpe_File* is described in the *mpe_File* section.

value returned If the call is successful, *mpeos_fileOpen()* should return MPE_FS_ERROR_SUCCESS and a pointer to the file identifier in *fileHandle*. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE

indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NO_MOUNT

indicates no device mount was found to handle the path.

MPE_FS_ERROR_NOT_FOUND

indicates the file path could not be found.

MPE_FS_ERROR_OUT_OF_MEM

indicates memory could not be allocated from the system.

MPE_FS_ERROR_READ_ONLY

indicates a file or directory is read-only.

description mpeos_fileOpen() should create an access handle to the file specified by *fileName*.

If the file already exists when performing a create, then mpeos_fileOpen() should open the file.

related function(s)

mpeos_fileClose
mpeos_fileGetFStat
mpeos_fileRead
mpeos_fileSetChangeListener
mpeos_fileSetFStat
mpeos_fileSync
mpeos_fileWrite

mpeos_fileRead

reads data from an opened file.

syntax `mpe_FileError mpeos_fileRead(
 mpe_File fileHandle,
 uint32_t * count,
 void * buffer);`

| | | |
|---------------------|-------------------|--|
| parameter(s) | <i>fileHandle</i> | specifies an open file access handle. <i>fileHandle</i> is returned by a previous call to <code>mpeos_fileOpen()</code> . <code>mpe_File</code> is described in the <i>mpe_File</i> section. |
| | <i>count</i> | is both an input and output pointer to a byte count. When called, <i>count</i> points to the number of bytes to write. When the call returns, <i>count</i> indicates the number of bytes actually written. |
| | <i>buffer</i> | is an output pointer to the buffer (of minimum size <i>count</i>) to be filled with the read data from the file. |

value returned If the call is successful, `mpeos_fileRead()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`
indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_FS_ERROR_NOT_FOUND`
indicates the file path could not be found.

description `mpeos_fileRead()` should read data from an opened file access handle into a buffer. Up to *count* bytes are read into the *buffer*. The number of bytes actually read (which may be less than the number of bytes originally requested) are returned in *count*.

-
- ◆ **NOTE:** To read from the file, the file must be opened with `MPE_FS_OPEN_READ` permission.
-

related function(s) `mpeos_fileClose`
`mpeos_fileGetFStat`
`mpeos_fileOpen`
`mpeos_fileSetChangeListener`
`mpeos_fileSetFStat`
`mpeos_fileWrite`

mpeos_fileRename

renames a file.

syntax mpe_FileError mpeos_fileRename(
 const char * *oldName*,
 const char * *newName*);

parameter(s) *oldName* is an input pointer to the path of an existing file to rename or move.

newName is an input pointer to the new path and/or name of the file.

value returned If the call is successful, mpeos_fileRename should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE
 indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER
 indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NO_MOUNT
 indicates no device mount was found to handle the path.

MPE_FS_ERROR_NOT_FOUND
 indicates the file path could not be found.

MPE_FS_ERROR_OUT_OF_MEM
 indicates memory could not be allocated from the system.

description mpeos_fileRename() should rename and/or move the file.

-
- ◆ **NOTE:** Both the old and new file names must exist in the same file-system device. Individual file systems may also impose additional restrictions, such as not allowing a file to be renamed into a different directory.
-

related function(s) mpeos_dirDelete
 mpeos_fileDelete

mpeos_fileRemoveChangeListener

removes an event listener for the specified file.

syntax mpe_FileError mpeos_fileRemoveChangeListener(
 mpe_FileChangeHandle *handle*);

parameter(s) *fileHandle* specifies an open file access handle. *fileHandle* is returned by a previous call to `mpeos_fileSetChangeListener()`.
mpe_FileChangeHandle is described in the *mpe_FileChangeHandle* section.

value returned If the call is successful, `mpeos_fileRemoveChangeListener()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`
indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_FS_ERROR_NOT_FOUND`
indicates the file path could not be found.

`MPE_FS_ERROR_UN SUPPORT`
indicates the requested information *mode* is not currently supported for the file.

description `mpeos_fileSetRemoveChangeListener()` remove the change listener associated with the specified file handle.

related function(s) `mpeos_fileSetChangeListener`

mpeos_fileSetChangeListener

registers a listener.

```
syntax mpe_FileError mpeos_fileSetChangeListener(
    const char * fileName,
    mpe_EventQueue queueId ,
    void * act,
    mpe_FileChangeHandle * handle );
```

| | |
|---------------------|--|
| parameter(s) | <p><i>fileName</i> is an input pointer to the file to monitor for changes. <i>fileName</i> is returned by a previous call to <code>mpeos_fileOpen()</code>.</p> <p><i>queueId</i> identifies the queue in which to send the new version number for the changed file. <code>mpe_EventQueue</code> is described in the <i>mpe_EventQueue</i> section.</p> <p><i>act</i> is an input pointer to an Asynchronous Completion Token (ACT). When events are sent to the event queue specified in (via <code>mpeos_eventQueueSend()</code>), the event should pass this ACT pointer in the <i>optionalEventData2</i> field.</p> <p><i>fileHandle</i> is an input pointer to an open file access handle. <i>fileHandle</i> is returned by a previous call to <code>mpeos_fileOpen()</code>. <code>mpe_FileChangeHandle</code> is described in the <i>mpe_FileChangeHandle</i> section.</p> |
|---------------------|--|

value returned If the call is successful, `mpeos_fileSetChangeListener()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`
 indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`
 indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_fileSetChangeListener()` should register a listener for changes made to the indicated file.

related function(s) `mpeos_eventQueueSend`
`mpeos_fileRemoveChangeListener`

mpeos_fileSetFStat

sets information about a file based on its open handle.

syntax `mpe_FileError mpeos_fileSetFStat(mpe_File fileHandle, mpe_FileStatMode mode, mpe_FileInfo * info);`

| | | |
|---------------------|--|--|
| parameter(s) | <i>fileHandle</i> | specifies an open file access handle. <i>fileHandle</i> is returned by a previous call to <code>mpeos_fileOpen()</code> . <code>mpe_File</code> is described in the <i>mpe_File</i> section. |
| | <i>mode</i> | specifies the information being requested. <code>mpe_FileStatMode</code> is described in the <i>mpe_FileStatMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | <code>MPE_FS_STAT_EXPDATE</code> | sets the expiration date. |
| | <code>MPE_FS_STAT_OBJCHANGE_OFF</code> | un-registers for object change updates. |
| | <code>MPE_FS_STAT_OBJCHANGE_ON</code> | registers for object change updates. |
| | <code>MPE_FS_STAT_ORG_ACCESS</code> | sets the list of organization IDs that have read and/or write access. |
| | <code>MPE_FS_STAT_PERM</code> | sets the permission for a Unix style file. |
| | <code>MPE_FS_STAT_PRIOR</code> | sets the priority of the file. |
| | <code>MPE_FS_STAT_SIZE</code> | sets the size of the file. |
| <i>info</i> | | is an input pointer to the buffer with the requested information to update. <code>mpe_FileInfo</code> is described in the <i>mpe_FileInfo</i> section. |

value returned If the call is successful, `mpeos_fileSetFStat()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

- `MPE_FS_ERROR_DEVICE_FAILURE`
indicates the underlying call failed.
- `MPE_FS_ERROR_INVALID_PARAMETER`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NOT_FOUND

indicates the file path could not be found.

MPE_FS_ERROR_UNSUPPORT

indicates the requested information *mode* is not currently supported for the file.

description mpeos_fileSetFStat() should set information in a previously opened file.

related function(s)

mpeos_fileClose

mpeos_fileGetFStat

mpeos_fileGetStat

mpeos_fileOpen

mpeos_fileSetStat

mpeos_fileSetStat

sets information about a file based on its pathname.

syntax `mpe_FileError mpeos_fileSetStat(const char * fileName, mpe_FileStatMode mode, mpe_FileInfo * info);`

parameter(s) *fileName* is an input pointer to the path of an existing file in which to set the specified information.

mode specifies the information being requested. *mpe_FileStatMode* is described in the *mpe_FileStatMode* section. Currently, the following values are supported for *mode*:

`MPE_FS_STAT_EXPDATE`
sets the expiration date.

`MPE_FS_STAT_OBJCHANGE_OFF`
un-registers for object change updates.

`MPE_FS_STAT_OBJCHANGE_ON`
registers for object change updates.

`MPE_FS_STAT_ORG_ACCESS`
sets the list of organization IDs that have read and/or write access.

`MPE_FS_STAT_PERM`
sets the permission for a Unix style file.

`MPE_FS_STAT_PRIOR`
sets the priority of the file.

`MPE_FS_STAT_SIZE`
sets the size of the file.

info is an input pointer to the buffer with the requested information to update. *mpe_FileInfo* is described in the *mpe_FileInfo* section.

value returned If the call is successful, `mpeos_fileSetStat()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`
indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_FS_ERROR_NO_MOUNT`
indicates no device mount was found to handle the path.

MPE_FS_ERROR_NOT_FOUND

indicates the file path could not be found.

MPE_FS_ERROR_OUT_OF_MEM

indicates memory could not be allocated from the system.

MPE_FS_ERROR_UNSUPPORT

indicates the requested information *mode* is not currently supported for the file.

description mpeos_fileSetStat() should update information in the file specified by *fileName*.

related function(s)

mpeos_fileClose

mpeos_fileGetFStat

mpeos_fileGetStat

mpeos_fileOpen

mpeos_fileSetFStat

mpeos_fileSync

synchronizes file contents on the storage device with any pending file data.

syntax mpe_FileError mpeos_fileSync(mpe_File *fileHandle*);

parameter(s) *fileHandle* specifies an open file access handle. *fileHandle* is returned by a previous call to mpeos_fileOpen(). *mpe_File* is described in the *mpe_File* section.

value returned If the call is successful, mpeos_fileSync() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_FS_ERROR_DEVICE_FAILURE

indicates the underlying call failed.

MPE_FS_ERROR_INVALID_PARAMETER

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

MPE_FS_ERROR_NOT_FOUND

indicates the file path could not be found.

description mpeos_fileSync() should synchronize the file contents of an opened file access handle, flushing any unwritten data cached in memory out to the underlying file system.

related function(s) mpeos_fileClose
mpeos_fileOpen
mpeos_fileWrite

mpeos_fileWrite

writes data to an open file.

syntax `mpe_FileError mpeos_fileWrite(
 mpe_File fileHandle,
 uint32_t * count,
 void * buffer);`

| | | |
|---------------------|-------------------|--|
| parameter(s) | <i>fileHandle</i> | specifies an open file access handle. <i>fileHandle</i> is returned by a previous call to <code>mpeos_fileOpen()</code> . <code>mpe_File</code> is described in the <i>mpe_File</i> section. |
| | <i>count</i> | is both an input and output pointer to a byte count. When called, <i>count</i> must point to the number of bytes to write. When the call returns, <i>count</i> indicates the number of bytes actually written. |
| | <i>buffer</i> | is an input pointer to the data buffer (of minimum size <i>count</i>) to be written to the file. |

value returned If the call is successful, `mpeos_fileWrite()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_FS_ERROR_DEVICE_FAILURE`
indicates the underlying call failed.

`MPE_FS_ERROR_INVALID_PARAMETER`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_FS_ERROR_NOT_FOUND`
indicates the file path could not be found.

description `mpeos_fileWrite()` should write data to an opened file access handle from a buffer at the current handle's read/write position.

- ◆ **NOTE:** To write to a file, the file must be opened with `MPE_FS_OPEN_WRITE` permission. Also, some underlying file-systems (for example, ROMFS, object carousel, etc.) are read-only and do not support writing to files.

related function(s) `mpeos_fileClose`
`mpeos_fileOpen`
`mpeos_fileRead`
`mpeos_fileSetChangeListener`
`mpeos_fileSync`

mpeos_init

initializes this MPEOS operating-system specific file-system interface.

syntax void mpeos_init(const char * *mount*);

parameter(s) *mount* is an input pointer to the operating-system specific mount.

value returned none

description mpeos_init() should initialize the file-system API, including the mounting of all MPEOS file-system root devices.

related function(s) none

PER File Persistent API

Overview

The file persistent Application Programming Interface (API) functions are platform independent and do not need to be ported. These functions provide support to other ported MPEOS functions. OCAP requires file attributes that are not supported by most file systems -- these functions give porters a common way to incorporate these attributes into their file-system implementations.

Before reading this chapter, you should be:

- ◆ familiar with file systems in general
- ◆ knowledgeable of the functions for any file systems provided natively by the operating system

After reading this chapter, you should be more knowledgeable of the OCAP stack

Error codes

The following error codes, which are defined in `mpe_file.h`, are used by the file persistent functions:

```
MPE_FS_ERROR_INVALID_PARAMETER  MPE_FS_ERROR_OUT_OF_MEM  
MPE_FS_ERROR_SUCCESS
```

MPE_FS_ERROR_INVALID_PARAMETER

`MPE_FS_ERROR_INVALID_PARAMETER` indicates an invalid parameter was specified. `MPE_FS_ERROR_INVALID_PARAMETER` is defined as follows:

```
#define MPE_FS_ERROR_INVALID_PARAMETER (MPE_FS_ERROR_BASE + 5)
```

MPE_FS_ERROR_OUT_OF_MEM

`MPE_FS_ERROR_OUT_OF_MEM` indicates memory could not be allocated. `MPE_FS_ERROR_OUT_OF_MEM` is defined as follows:

```
#define MPE_FS_ERROR_OUT_OF_MEM (MPE_FS_ERROR_BASE + 4)
```

MPE_FS_ERROR_SUCCESS

`MPE_FS_ERROR_SUCCESS` indicates a call was successful. `MPE_FS_ERROR_SUCCESS` is defined as follows:

```
#define MPE_FS_ERROR_SUCCESS (MPE_SUCCESS)
```

Data types and structures

The following data types and structures, which are defined in `mpe_file.h`, `mpe_time.h`, and `mpeos_file.h`, are used by the file persistent functions:

- ◆ **NOTE:** `char` type strings are required to be encoded in the Unicode UTF-8 character format.

| | |
|-------------------------------------|-----------------------|
| <code>mpe_FileError</code> | <code>mpe_Time</code> |
| <code>mpeos_filesys_ftable_t</code> | |

`mpe_FileError`

`mpe_FileError` returns error codes from the functions on the MPE file system. `mpe_FileError` is defined in `mpe_file.h` as follows:

```
typedef int32 mpe_FileError;
```

`mpe_Time`

`mpe_Time` specifies the time in seconds. `mpe_Time` is defined in `mpe_time.h` as follows:

```
typedef os_Time mpe_Time;
```

`mpeos_filesys_ftable_t`

`mpeos_filesys_ftable_t` is the function table for individual drivers. The file system module calls into these functions when performing operations. Each file-system driver (which could be multiple drivers) must implement a set of functions, each with a specific parameter list and functionality as appropriate for that file system. After the functions are implemented, a `mpeos_filesys_ftable_t` is created referencing those file system functions. Unlike other modules that have specific functions to implement, the file system has functions to implement for each driver.

`mpeos_filesys_ftable_t` is defined in `mpeos_file.h` as follows:

```
typedef struct mpeos_filesys_ftable_t {
    void (*mpeos_init_ptr)(const char* mountPoint);
    mpe_FileError (*mpeos_fileOpen_ptr)(const char* fileName,
                                         mpe_FileOpenMode openMode, mpe_File* returnHandle);
    mpe_FileError (*mpeos_fileClose_ptr)(mpe_File handle);
    mpe_FileError (*mpeos_fileRead_ptr)(mpe_File handle,
                                         uint32_t* count, void* buffer);
    mpe_FileError (*mpeos_fileWrite_ptr)(mpe_File handle,
                                         uint32_t* count, void* buffer);
    mpe_FileError (*mpeos_fileSeek_ptr)(mpe_File handle,
                                         mpe_FileSeekMode seekMode, int64* offset);
    mpe_FileError (*mpeos_fileSync_ptr)(mpe_File handle);
    mpe_FileError (*mpeos_fileGetStat_ptr)(
        const char* fileName, mpe_FileStatMode mode,
        mpe_FileInfo *info);
    mpe_FileError (*mpeos_fileSetStat_ptr)(
        const char* fileName, mpe_FileStatMode mode,
        mpe_FileInfo *info);
    mpe_FileError (*mpeos_fileGetFStat_ptr)(mpe_File handle,
                                             mpe_FileStatMode mode, mpe_FileInfo *info);
}
```

```

mpe_FileError (*mpeos_fileSetFStat_ptr)(mpe_File handle,
    mpe_FileStatMode mode, mpe_FileInfo *info);
mpe_FileError (*mpeos_fileDelete_ptr)(
    const char* fileName);
mpe_FileError (*mpeos_fileRename_ptr)(const char* oldName,
    const char* newName);
mpe_FileError (*mpeos_fileLoad_ptr)(const char *fileName,
    mpe_EventQueue , void *act, int cacheMode);
mpe_FileError (*mpeos_fileUnload_ptr)(
    const char *fileName);
mpe_FileError (*mpeos_dirOpen_ptr)(const char* name,
    mpe_Dir* returnHandle);
mpe_FileError (*mpeos_dirRead_ptr)(mpe_Dir handle,
    mpe_DirEntry* dirEnt);
mpe_FileError (*mpeos_dirClose_ptr)(mpe_Dir handle);
mpe_FileError (*mpeos_dirDelete_ptr)(const char* dirName);
mpe_FileError (*mpeos_dirRename_ptr)(const char* oldName,
    const char* newName);
mpe_FileError (*mpeos_dirCreate_ptr)(const char* dirName);
mpe_FileError (*mpeos_dirMount_ptr)(
    const mpe_DirUrl *dirUrl);
mpe_FileError (*mpeos_dirUnmount_ptr)(
    const mpe_DirUrl *dirUrl);
mpe_FileError (*mpeos_dirGetUStat_ptr)(
    const mpe_DirUrl *dirUrl, mpe_DirStatMode mode,
    mpe_DirInfo *info);
mpe_FileError (*mpeos_dirSetUStat_ptr)(
    const mpe_DirUrl *dirUrl, mpe_DirStatMode mode,
    mpe_DirInfo *info);
mpe_FileError (*mpeos_streamOpen_ptr) (
    const char *fileName, mpe_Stream *streamHandlePtr);
mpe_FileError (*mpeos_streamClose_ptr)(
    mpe_Stream streamHandle);
mpe_FileError (*mpeos_streamReadEvent_ptr)(
    mpe_Stream streamHandle, mpe_StreamEventInfo *event);
mpe_FileError (*mpeos_streamGetNumTaps_ptr)(
    mpe_Stream streamHandle, uint16_t tapType,
    uint32_t *numTaps);
mpe_FileError (*mpeos_streamReadTap_ptr)(
    mpe_Stream streamHandle, uint16_t tapType,
    uint32_t tapNumber, uint16_t *tap, uint16_t *tapID);
mpe_FileError (*mpeos_filePrefetch_ptr)(
    const char *fileName);
mpe_FileError (*mpeos_filePrefetchModule_ptr)(
    const char *mountpoint, const char *moduleName);
mpe_FileError (*mpeos_fileAddDII_ptr)(
    const char *mountpoint, uint16_t diiId, uint16_t
assocTag);
} mpeos_filesys_ftable_t;

```

Supported functions

The following file persistent functions need to be supported:

`mpeos_persistentGetStatFileName`
gets the status of a file.

`mpeos_persistentGetTempStatFileName`
gets the name of the temporary status file.

`mpeos_persistentFileInfoDelete`
deletes the temporary status file.

`mpeos_persistentFileInfoRename`
renames the temporary status file.

`mpeos_persistentSetDefaultFileAttributes`
sets the default values for file attributes.

-
- ◆ **NOTE:** char type strings are required to be encoded in the Unicode UTF-8 character format.
-

mpeos_persistentGetStatFileName

gets the status of a file.

syntax mpe_FileError mpeos_persistentGetStatFileName(
 const char * *filePath*,
 char ** *statFilePath*,
 char *magicChar*);

parameter(s) *filePath* is an input pointer to the file path.
statFilePath is an output pointer to the file status.
magicChar specifies the magic charcoal.

value returned If the call is successful, mpeos_persistentGetStatFileName() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return the following error code:

 MPE_FS_ERROR_OUT_OF_MEM
 indicates memory could not be allocated from the system.

description mpeos_persistentGetStatFileName() should get the status of file specified by *filePath*. This gets the name of the temporary status file used to preserve the status file integrity. It is the original status file appended with an additional *magicChar*.

-
- ◆ **NOTE:** The caller is responsible for freeing memory allocated to the *statFilePath* string.
-

related function(s) none

mpeos_persistentGetTempStatFileName

gets the name of the temporary status file.

syntax mpe_FileError mpeos_persistentGetTempStatFileName(
 const char * *filePath*,
 char ** *tempStatFilePath*,
 char *magicChar*);

parameter(s) *filePath* is an input pointer to the file path.

tempStatFilePath is an output pointer to the temporary file path.

magicChar specifies the magic charcoal.

value returned If the call is successful, mpeos_persistentGetTempStatFileName() should return MPE_FS_ERROR_SUCCESS. Otherwise, it should return the following error code:

MPE_FS_ERROR_OUT_OF_MEM

indicates memory could not be allocated from the system.

description mpeos_persistentGetTempStatFileName() should get the name of the temporary status file used to preserve the status file integrity. It is the original status file appended with an additional *magicChar*.

-
- ◆ **NOTE:** The caller is responsible for freeing memory allocated to the *tempStatFilePath* string.
-

related function(s) none

mpeos_persistentFileInfoDelete

deletes the temporary status file.

syntax mpe_Error mpeos_persistentFileInfoDelete(
 mpeos_filesys_ftable_t * *ftable*,
 const char * *filePath*,
 char *magicChar*);

parameter(s) *ftable* is an input pointer to the function table.
mpeos_filesys_ftable_t is defined in the
mpeos_filesys_ftable_t section.

filePath is an input pointer to the file path.

magicChar specifies the magic charcoal.

value returned If the call is successful, *mpeos_persistentFileInfoDelete()* should return *MPE_FS_ERROR_SUCCESS*. Otherwise, it should return the following error code:

MPE_FS_ERROR_INVALID_PARAMETER
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description *mpeos_persistentFileInfoDelete()* should delete the temporary status file.

related function(s) none

mpeos_persistentFileInfoRename

renames the temporary status file.

syntax `mmpe_FileError mpeos_persistentFileInfoRename(mpeos_filesys_ftable_t * ftable, const char * oldFilePath, const char * newFilePath, char magicChar);`

| | | |
|---------------------|--------------------|---|
| parameter(s) | <i>ftable</i> | is an input pointer to the function table. <i>mpeos_filesys_ftable_t</i> is defined in the <i>mpeos_filesys_ftable_t</i> section. |
| | <i>oldFilePath</i> | is an input pointer to the old file path. |
| | <i>newFilePath</i> | is an input pointer to the new file path. |
| | <i>magicChar</i> | specifies the magic charcoal. |

value returned If the call is successful, `mpeos_persistentFileInfoRename()` should return `MPE_FS_ERROR_SUCCESS`. Otherwise, it should return the following error code:

`MPE_FS_ERROR_INVALID_PARAMETER`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_persistentFileInfoRename()` should rename a temporary status file.

related function(s) none

mpeos_persistentSetDefaultFileAttributes

sets the default values for file attributes.

syntax void mpeos_persistentSetDefaultFileAttributes(
 uint16_t * permissions,
 uint8_t * priority,
 mpe_Time * expDate,
 int32 * numReadAccessOrgs,
 uint16_t ** readAccessOrgs,
 int32 * numWriteAccessOrgs,
 uint16_t ** writeAccessOrgs);

| | | |
|----------------------------|--|---|
| parameter(s) | <i>permissions</i> | is an input pointer to the permissions. |
| | <i>priority</i> | is an input pointer to the priority. |
| | <i>expDate</i> | is an input pointer to the expiration date. <i>mpe_Time</i> is defined in the <i>mpe_Time</i> section. |
| | <i>numReadAccessOrgs</i> | is an input pointer to the number of organization IDs allowed with read access. OCAP applications have an assigned organization ID based on the organization that wrote and delivered the application. File access may be limited to certain organizations. |
| | <i>readAccessOrgs</i> | is an input pointer to a dynamically allocated array of organization IDs with read access. |
| | <i>numWriteAccessOrgs</i> | is an input pointer to the number of organization IDs allowed with read access. |
| | <i>writeAccessOrgs</i> | is an input pointer to a dynamically allocated array of organization IDs with write access. |
| value returned | none | |
| description | mpeos_persistentSetDefaultFileAttributes() should assign the default value (as specified by MHP/OCAP) for each of the file attributes. | |
| related function(s) | none | |

FPL Front Panel API

Overview

The front panel Application Programming Interface (API) provides support for an Xlet with MonitorAppPermission to control the front panel indicators and text display of a set-top box. The front panel text display may be configured to display network time or a custom text string. Static front panel indicators (for example, a “You’ve got mail” icon) may also be lit or unlit by the implementation.

Before reading this chapter, you should be familiar with:

- ◆ what the front panel functions are and how they work
- ◆ the optional front panel display requirements described in the OCAP Front Panel Extension Specification

After reading this chapter, you should be familiar with the Front Panel Extension within the OCAP 1.0 stack, as implemented by Vidiom Systems, Inc.

Definitions

The front panel LEDS may be controlled for color, brightness, and to output specific text or pre-defined messages. The following front panel definitions are defined in `mpeos_frontpanel.h` as follows:

| | |
|--|--------------------------------------|
| <code>MPE_FP_BRIGHTNESS_OFF</code> | <code>MPE_FP_BRIGHTNESS_FULL</code> |
| <code>MPE_FP_COLOR_BLUE</code> | <code>MPE_FP_COLOR_GREEN</code> |
| <code>MPE_FP_COLOR_ORANGE</code> | <code>MPE_FP_COLOR_RED</code> |
| <code>MPE_FP_COLOR_UNSUPPORTED</code> | <code>MPE_FP_COLOR_YELLOW</code> |
| <code>MPE_FP_INDICATOR_MESSAGE</code> | <code>MPE_FP_INDICATOR_POWER</code> |
| <code>MPE_FP_INDICATOR_RECORD</code> | <code>MPE_FP_INDICATOR_REMOTE</code> |
| <code>MPE_FP_INDICATOR_RFBYPASS</code> | <code>MPE_FP_INDICATOR_TEXT</code> |

`MPE_FP_BRIGHTNESS_OFF`

`MPE_FP_BRIGHTNESS_OFF` specifies the minimum brightness level, which is off. `MPE_FP_BRIGHTNESS_OFF` is defined as follows:

```
#define MPE_FP_BRIGHTNESS_OFF 0
```

`MPE_FP_BRIGHTNESS_FULL`

`MPE_FP_BRIGHTNESS_FULL` specifies the maximum possible brightness level. The actual maximum brightness for a specific platform is implementation dependent and must be between 1 and `MPE_FP_BRIGHTNESS_FULL`.

`MPE_FP_BRIGHTNESS_FULL` is defined as follows:

```
#define MPE_FP_BRIGHTNESS_FULL 127
```

- ◆ **NOTE:** There must be a minimum of two brightness levels: 0 (specifies off) and 1 (specifies on). A negative value for the brightness setting is illegal.

`MPE_FP_COLOR_BLUE`

`MPE_FP_COLOR_BLUE` specifies the color blue bit field. `MPE_FP_COLOR_BLUE` is defined as follows:

```
#define MPE_FP_COLOR_BLUE 0x01
```

`MPE_FP_COLOR_GREEN`

`MPE_FP_COLOR_GREEN` specifies the color green bit field.

`MPE_FP_COLOR_GREEN` is defined as follows:

```
#define MPE_FP_COLOR_GREEN 0x02
```

`MPE_FP_COLOR_ORANGE`

`MPE_FP_COLOR_ORANGE` specifies the color orange bit field.

`MPE_FP_COLOR_ORANGE` is defined as follows:

```
#define MPE_FP_COLOR_ORANGE 0x10
```

MPE_FP_COLOR_RED

MPE_FP_COLOR_RED specifies the color red bit field. MPE_FP_COLOR_RED is defined as follows:

```
#define MPE_FP_COLOR_RED 0x04
```

MPE_FP_COLOR_UNSUPPORTED

MPE_FP_COLOR_UNSUPPORTED specifies that color support is not available for this implementation bit field. MPE_FP_COLOR_UNSUPPORTED is defined as follows:

```
#define MPE_FP_COLOR_UNSUPPORTED 0x00
```

MPE_FP_COLOR_YELLOW

MPE_FP_COLOR_YELLOW specifies the color yellow bit field.
MPE_FP_COLOR_YELLOW is defined as follows:

```
#define MPE_FP_COLOR_YELLOW 0x08
```

MPE_FP_INDICATOR_MESSAGE

MPE_FP_INDICATOR_MESSAGE specifies the standard message indicator.
MPE_FP_INDICATOR_MESSAGE is defined as follows:

```
#define MPE_FP_INDICATOR_MESSAGE "message"
```

MPE_FP_INDICATOR_POWER

MPE_FP_INDICATOR_POWER specifies the standard power indicator.
MPE_FP_INDICATOR_POWER is defined as follows:

```
#define MPE_FP_INDICATOR_POWER "power"
```

MPE_FP_INDICATOR_RECORD

MPE_FP_INDICATOR_RECORD specifies the standard record indicator.
MPE_FP_INDICATOR_RECORD is defined as follows:

```
#define MPE_FP_INDICATOR_RECORD "record"
```

MPE_FP_INDICATOR_REMOTE

MPE_FP_INDICATOR_REMOTE specifies the standard remote indicator.
MPE_FP_INDICATOR_REMOTE is defined as follows:

```
#define MPE_FP_INDICATOR_REMOTE "remote"
```

MPE_FP_INDICATOR_RFBYPASS

MPE_FP_INDICATOR_RFBYPASS specifies the standard RF bypass indicator.
MPE_FP_INDICATOR_RFBYPASS is defined as follows:

```
#define MPE_FP_INDICATOR_RFBYPASS "rfbypass"
```

MPE_FP_INDICATOR_TEXT

MPE_FP_INDICATOR_TEXT specifies the standard text display. This indicator corresponds to the front panel text display. MPE_FP_INDICATOR_TEXT is defined as follows:

```
#define MPE_FP_INDICATOR_TEXT "text"
```

Error codes

The following error codes are supported for the module support API. They are defined in `mpe_error.h` as follows:

MPE_EINVAL

MPE_SUCCESS

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

MPE_EINVAL

MPE_EINVAL indicates at least one input parameter to the function has an invalid value. MPE_EINVAL is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_SUCCESS

MPE_SUCCESS indicates successful completion of an `mpe_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). MPE_SUCCESS is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types are in `mpeos_frontpanel.h` as follows:

| | |
|-------------------------------|----------------------------------|
| <code>mpe_FpBlinkSpec</code> | <code>mpe_FpCapabilities</code> |
| <code>mpe_FpScrollSpec</code> | <code>mpe_FpTextPanelMode</code> |

`mpe_FpBlinkSpec`

`mpe_FpBlinkSpec` specifies how the front panel text or indicator blinks.
`mpe_FpBlinkSpec` is defined as follows:

```
typedef struct mpe_FpBlinkSpec {
    uint16_t iterations;
    uint16_t onDuration;
} mpe_FpBlinkSpec;
```

where

`iterations` specifies the number of times per minute for the text display or indicator to blink. 0 indicates no blinking.

`onDuration` specifies the percentage of time per blink the LED is on. Values range from 0 to 100. 0 indicates the LED should never be on and effectively turns off the display. 100 indicates the LED should never be off and effectively puts the display into a non-blinking, continuously on, mode.

`mpe_FpCapabilities`

`mpe_FpCapabilities` defines the capability set of the front panel display.
`mpe_FpCapabilities` is defined as follows:

```
typedef struct mpe_FpCapabilities {
    uint32_t totalIndicators;
    char** indicatorNames;
    uint32_t* colors;
    uint32_t* brightness;
    uint32_t* maxCycleRate;
    char* supportedChars;
    uint32_t columns;
    uint32_t rows;
    uint32_t maxHorizontalIterations;
    uint32_t maxVerticalIterations;
} mpe_FpCapabilities;
```

where

`totalIndicators`

represents the total number of unique indicators available on the front panel display, including a text panel, if available.

`indicatorNames`

is a pointer to an array of indicator names. The array is of length `totalIndicators`. Each name is a null-terminated string which specifies the unique name of that indicator. The OCAP Front Panel Extension API mandates the use of certain indicator names if they are supported by the underlying hardware. The indicator are defined in the *Definitions* section (`MPE_FP_INDICATOR_xxx`).

| | |
|--------------------------------------|--|
| <code>colors</code> | is a pointer to an array of integers, corresponding to each indicator in <code>indicatorNames</code> , set to the bitwise-OR of all display colors available to each specific indicator. |
| <code>brightness</code> | is a pointer to an array of integers, corresponding to each indicator in <code>indicatorNames</code> , set to the number of discrete brightness settings available to each indicator. Values range from 2 to 128. |
| <code>maxCycleRate</code> | is a pointer to an array of integers, corresponding to each indicator in <code>indicatorNames</code> , which specify the maximum number of times per minute each indicator can blink. Zero indicates blinking is not supported. |
| <code>supportedChars</code> | is a pointer to an array of all characters that can be displayed on the text panel. <code>NULL</code> if there is no text panel support on this host. |
| <code>columns</code> | specifies the number of characters that can be displayed on each row of the text panel. |
| <code>rows</code> | specifies the number of rows available for text display on the text panel. |
| <code>maxHorizontalIterations</code> | specifies the maximum number of times per minute that characters can scroll right-to-left across the text display with zero hold time set. Zero indicates scrolling is not supported. For details on scrolling, refer to the <i>mpe_FpScrollSpec</i> section. |
| <code>maxVerticalIterations</code> | specifies the maximum number of times per minute that rows of text can scroll bottom-to-top across the text display with zero hold time set. Zero indicates vertical scrolling is not supported. For details on scrolling, refer to the <i>mpe_FpScrollSpec</i> section. |

mpe_FpScrollSpec

`mpe_FpScrollSpec` specifies how the front panel display text scrolls.

`mpe_FpScrollSpec` is defined as follows:

```
typedef struct mpe_FpScrollSpec {
    int16 horizontalIterations;
    int16 verticalIterations;
    int16 holdDuration;
} mpe_FpScrollSpec;
```

where

`horizontalIterations`

specifies the number of times per minute the characters on the display scroll across the display from right to left. 0 indicates horizontal scrolling is disabled. -1 indicates more than one row displays and the characters scroll vertically.

verticalIterations

specifies the number of times per minute the rows of text on the display scroll across the display from bottom to top. 0 indicates vertical scrolling is disabled. -1 indicates the display only supports a single row of characters and text always scrolls horizontally.

holdDuration indicates the percentage of time the display holds at each character, or line, while scrolling. Values range from 0 to 100.

mpe_FpTextPanelMode

mpe_FpTextPanelMode specifies the operational mode of the front text panel. **mpe_FpTextPanelMode** is defined as follows:

```
typedef enum mpe_FpTextPanelMode {
    MPE_FP_TEXT_MODE_CLOCK_12HOUR = 0x00,
    MPE_FP_TEXT_MODE_CLOCK_24HOUR = 0x01,
    MPE_FP_TEXT_MODE_STRING = 0x02
} mpe_FpTextPanelMode;
```

where

MPE_FP_TEXT_MODE_CLOCK_12HOUR

displays the network time using a standard 12 hour HH:MM format.

MPE_FP_TEXT_MODE_CLOCK_24HOUR

displays the network time using a standard 24 hour HH:MM format.

MPE_FP_TEXT_MODE_STRING

displays a custom string of characters.

Supported functions

The following front panel functions need to be provided:

- `mpeos_fpGetCapabilities`
gets the capabilities of the front panel.
- `mpeos_fpGetIndicator`
gets the current settings for a specified indicator.
- `mpeos_fpGetText`
gets the mode and settings for the front panel text display.
- `mpeos_fpInit` initializes platform specific, front-panel support.
- `mpeos_fpSetIndicator`
sets the specified indicator to the given values.
- `mpeos_fpSetText`
sets the mode and settings for the front panel text display.

mpeos_fpGetCapabilities

gets the capabilities of the front panel.

syntax mpe_Error mpeos_fpGetCapabilities(
 mpe_FpCapabilities ** *capabilities*);

parameter(s) *capabilities* is an input pointer to a pointer to the location of an *mpe_FpCapabilities* structure describing the capabilities of the front panel display. This capabilities structure is usually allocated by the system-level software and is not altered by the stack. A pointer can be set to point to a constant structure. *mpe_FpCapabilities* is described in the *mpe_FpCapabilities* section.

value returned If the call is successful, *mpeos_fpGetCapabilities()* should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *capabilities* has an invalid value.

description *mpeos_fpGetCapabilities()* should get the capabilities of the front panel display.

related function(s) *mpeos_fpGetIndicator*
mpeos_fpGetText
mpeos_fpSetIndicator
mpeos_fpSetText

mpeos_fpGetIndicator

gets the current settings for a specified indicator.

syntax `mpe_Error mpeos_fpGetIndicator(uint32_t indicator, uint32_t * brightness, uint32_t * color, mpe_FpBlinkSpec * blinkSpec);`

| | | |
|-----------------------|-------------------------|---|
| parameter(s) | <i>indicator</i> | specifies the indicator of interest. The number corresponds to the index into the indicator array provided by <code>mpeos_fpGetCapabilities()</code> . The text panel should not be specified through this routine (<code>MPE_FP_INDICATOR_TEXT</code>). |
| | <i>brightness</i> | is an output pointer to the location to hold the returned brightness level of the indicator. A value of <code>MPE_FP_BRIGHTNESS_OFF</code> means the indicator is off. Values above <code>MPE_FP_BRIGHTNESS_OFF</code> specify increasing levels of brightness up to the maximum indicator brightness determined by <code>mpeos_fpGetCapabilities.brightness[indicator]-1</code> . <code>MPE_FP_BRIGHTNESS_OFF</code> is described in the <code>MPE_FP_BRIGHTNESS_OFF</code> section. |
| | <i>color</i> | is an output pointer to the location to hold the returned color for the text display. The supported colors for this text display are found in the <code>colors</code> field of <code>mpe_FpCapabilities</code> at the index for <code>MPE_FP_INDICATOR_TEXT</code> . Currently, the following values are supported for <i>color</i> : |
| | | <code>MPE_FP_COLOR_BLUE</code> specifies the blue bit field. |
| | | <code>MPE_FP_COLOR_GREEN</code> specifies the green bit field. |
| | | <code>MPE_FP_COLOR_ORANGE</code> specifies the orange bit field. |
| | | <code>MPE_FP_COLOR_RED</code> specifies the red bit field. |
| | | <code>MPE_FP_COLOR_YELLOW</code> specifies the yellow bit field. |
| | <i>blinkSpec</i> | is an output pointer to the location to hold the returned value of the current blink settings for the indicator. <code>mpe_FpBlinkSpec</code> is described in the <code>mpe_FpBlinkSpec</code> section. |
| value returned | | If the call is successful, <code>mpeos_fpGetIndicator()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return the following error code: |
| | <code>MPE_EINVAL</code> | indicates <i>indicator</i> has an invalid value. |

description mpeos_fpGetIndicator() should get the current settings of the specified indicator.

related function(s) mpeos_fpGetCapabilities
mpeos_fpSetIndicator

mpeos_fpGetText

gets the mode and settings for the front panel text display.

syntax mpe_Error mpeos_fpGetText(
 mpe_FpTextPanelMode * mode,
 uint32_t * color,
 uint32_t * brightness,
 mpe_FpBlinkSpec * blinkSpec,
 mpe_FpScrollSpec * scrollSpec);

| | | |
|---------------------|-------------------------------|---|
| parameter(s) | <i>mode</i> | is an output pointer to the location to hold the returned text display mode. <i>mpe_FpTextPanelMode</i> is described in <i>mpe_FpTextPanelMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | MPE_FP_TEXT_MODE_CLOCK_12HOUR | displays the network time using a standard 12 hour HH:MM format. |
| | MPE_FP_TEXT_MODE_CLOCK_24HOUR | displays the network time using a standard 24 hour HH:MM format. |
| | MPE_FP_TEXT_MODE_STRING | displays a custom string of characters. |
| <i>color</i> | | is an output pointer to the desired color for the text display. The supported colors for this text display are found in the <i>colors</i> field of <i>mpe_FpCapabilities</i> at the index for MPE_FP_INDICATOR_TEXT. Currently, the following values are supported for <i>color</i> : |
| | MPE_FP_COLOR_BLUE | specifies the blue bit field. |
| | MPE_FP_COLOR_GREEN | specifies the green bit field. |
| | MPE_FP_COLOR_ORANGE | specifies the orange bit field. |
| | MPE_FP_COLOR_RED | specifies the red bit field. |
| | MPE_FP_COLOR_YELLOW | specifies the yellow bit field. |
| <i>brightness</i> | | is an output pointer to the brightness level for the text display. A value of MPE_FP_BRIGHTNESS_OFF means the display is off. Values above MPE_FP_BRIGHTNESS_OFF specify higher levels of brightness. MPE_FP_BRIGHTNESS_OFF is described in the <i>MPE_FP_BRIGHTNESS_OFF</i> section. |
| <i>blinkSpec</i> | | is an output pointer to the location to hold the blink settings for the text display. <i>mpe_FpBlinkSpec</i> is described in the <i>mpe_FpBlinkSpec</i> section. |

scrollSpec is an output pointer to the location to hold the front panel display text scroll settings. `mpe_FpScrollSpec` is described in the *mpe_FpScrollSpec* section.

value returned If the call is successful, `mpeos_fpGetText()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE EINVAL` indicates at least one parameter to the function has an invalid value.

description `mpeos_fpGetText()` should get the front panel text display mode and settings.

related function(s) `mpeos_fpGetCapabilities`
`mpeos_fpSetText`

mpeos_fpinit

initializes platform specific, front-panel support.

syntax mpeos_fpinit(void);

parameter(s) none

value returned none

description mpeos_fpinit() should initialize the platform specific, front-panel support.

related function(s) none

mpeos_fpSetIndicator

sets the specified indicator to the given values.

syntax `mpe_Error mpeos_fpSetIndicator(`
`uint32_t indicator,`
`uint32_t brightness,`
`uint32_t color,`
`mpe_FpBlinkSpec blinkSpec);`

| | | |
|-----------------------|-------------------------|---|
| parameter(s) | <i>indicator</i> | specifies the indicator settings to set. The number corresponds to the index into the indicator array provided by <code>mpeos_fpGetCapabilities()</code> . The text panel should not be specified through this routine. |
| | <i>brightness</i> | specifies the desired brightness level of the specified indicator. A value of <code>MPE_FP_BRIGHTNESS_OFF</code> means the indicator is off. Values above <code>MPE_FP_BRIGHTNESS_OFF</code> specify increasing levels of brightness up to the maximum indicator brightness determined by <code>mpeos_fpGetCapabilities.brightness[indicator]-1</code> . <code>MPE_FP_BRIGHTNESS_OFF</code> is described in the <i>MPE_FP_BRIGHTNESS_OFF</i> section. |
| | <i>color</i> | specifies the desired color for the text display. The supported colors for this text display are found in the <code>colors</code> field of <code>mpe_FpCapabilities</code> at the index for <code>MPE_FP_INDICATOR_TEXT</code> . Currently, the following values are supported for <code>color</code> : |
| | | <code>MPE_FP_COLOR_BLUE</code> specifies the blue bit field. |
| | | <code>MPE_FP_COLOR_GREEN</code> specifies the green bit field. |
| | | <code>MPE_FP_COLOR_ORANGE</code> specifies the orange bit field. |
| | | <code>MPE_FP_COLOR_RED</code> specifies the red bit field. |
| | | <code>MPE_FP_COLOR_YELLOW</code> specifies the yellow bit field. |
| | <i>blinkSpec</i> | specifies the desired blink settings for the indicator. <code>mpe_FpBlinkSpec</code> is described in the <i>mpe_FpBlinkSpec</i> section. |
| value returned | | If the call is successful, <code>mpeos_fpSetIndicator()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return the following error code: |
| | <code>MPE_EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |

description mpeos_fpSetIndicator() should set the specified indicator to the specified values.

related function(s) mpeos_fpGetCapabilities
mpeos_fpGetIndicator

mpeos_fpSetText

sets the mode and settings for the front panel text display.

```
syntax mpe_Error mpeos_fpSetText(
    mpe_FpTextPanelMode mode,
    uint32_t numTextLines,
    const char ** text,
    uint32_t color,
    uint32_t brightness,
    mpe_FpBlinkSpec blinkSpec,
    mpe_FpScrollSpec scrollSpec );
```

| | | |
|---------------------|--|--|
| parameter(s) | <i>mode</i> | specifies the mode to set for the text display as a selection from <code>mpe_FpTextPanelMode</code> . <code>mpe_FpTextPanelMode</code> is described in <i>mpe_FpTextPanelMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | <code>MPE_FP_TEXT_MODE_CLOCK_12HOUR</code> | displays the network time using a standard 12 hour HH:MM format. |
| | <code>MPE_FP_TEXT_MODE_CLOCK_24HOUR</code> | displays the network time using a standard 24 hour HH:MM format. |
| | <code>MPE_FP_TEXT_MODE_STRING</code> | displays a custom string of characters. |
| <i>numTextLines</i> | | specifies the number of lines of text to be displayed. This argument is ignored unless <code>MPE_FP_TEXT_MODE_STRING</code> is specified for <i>mode</i> . |
| <i>text</i> | | is an input pointer to an array (of length <i>numTextLines</i>) of null-terminated strings that make up the desired contents of the text display. The memory containing this text data is de-allocated when the function returns. This argument is ignored unless <code>MPE_FP_TEXT_MODE_STRING</code> is specified for <i>mode</i> . |
| <i>color</i> | | specifies the desired color for the text display. The supported colors for this text display are found in the <code>colors</code> field of <code>mpe_FpCapabilities</code> at the index for <code>MPE_FP_INDICATOR_TEXT</code> . Currently, the following values are supported for <i>color</i> : |
| | <code>MPE_FP_COLOR_BLUE</code> | specifies the blue bit field. |
| | <code>MPE_FP_COLOR_GREEN</code> | specifies the green bit field. |
| | <code>MPE_FP_COLOR_ORANGE</code> | specifies the orange bit field. |

| | |
|-------------------|--|
| | MPE_FP_COLOR_RED specifies the red bit field. |
| | MPE_FP_COLOR_YELLOW specifies the yellow bit field. |
| <i>brightness</i> | specifies the desired brightness level for the text display. A value of MPE_FP_BRIGHTNESS_OFF means the display is off. Values above MPE_FP_BRIGHTNESS_OFF specify higher levels of brightness up to and including the maximum level for the display found in the brightness field of <code>mpe_FpCapabilities</code> at the index for MPE_FP_INDICATOR_TEXT. MPE_FP_BRIGHTNESS_OFF is described in the <i>MPE_FP_BRIGHTNESS_OFF</i> section. |
| <i>blinkSpec</i> | specifies the desired blink settings for the text display. <code>mpe_FpBlinkSpec</code> is described in the <i>mpe_FpBlinkSpec</i> section. |
| <i>scrollSpec</i> | specifies the front panel display text scroll settings. <code>mpe_FpScrollSpec</code> is described in the <i>mpe_FpScrollSpec</i> section. |

value returned If the call is successful, `mpeos_fpSetText()` should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

description `mpeos_fpSetText()` should set the mode and other settings of the front panel text display.

related function(s) `mpeos_fpGetCapabilities`
`mpeos_fpGetText`

GFX Graphics Manager API

Overview

The Multimedia Platform Environment (MPE) graphics Application Programming Interface (API) contains the functions to implement the OpenCable Application Platform (OCAP) graphics model and APIs. It supports Java's Abstract Windowing Toolkit (AWT) graphics primitives, event dispatch, and fonts, as well as the Home Audio Video Interoperability (HAVi) device.

The MPE graphics package is designed to support the Java Virtual Machine (JVM)/AWT API and is part of the OCAP stack.

Before reading this chapter, you should be familiar with:

- ◆ the Java version that OCAP uses
- ◆ the AWT APIs used and the OCAP extensions to those APIs. For instance, `DVBGraphics`, `DVBA1phaComposite`, etc.
- ◆ the Porter-Duff drawing modes and the alpha modulation constant
- ◆ the OCAP system constants as defined in the OCAP specification

After reading this chapter, you should be:

- ◆ able to port the frame buffer interface and enables the MPE graphics layer

The MPE graphics package includes the following:

- ◆ MPE graphics manager APIs
- ◆ Multimedia Platform Environment Operating System (MPEOS) graphics abstraction porting layer

MPE graphics initialize the defined graphics API and the graphics manager when the system is turned on. MPE graphic functions are platform independent and are directly mapped to MPE operating system graphic functions.

The MPEOS graphics abstraction porting layer implements the MPE graphic API. Private functions of this layer make calls to the operating-system specific API; and therefore, need to be re-implemented if MPE graphics is ported to a different platform.

Graphic directory structure

Unlike the other header files in the OCAP stack, the graphic implementation is divided into separate header files depending on the functionality. Most of the header files are located in the OCAP-1.0\mpe\os\include\gfx directory. The graphic header files are as follows:

`mpeos_gfx.h` provides the general definitions, handles, and data structures used by all the graphic APIs. `mpeos_gfx.h` is located in the OCAP-1.0\mpe\os\include directory.

`ConvertUTF.h` provides the Unicode Transformation Format (UTF) schemes between UTF32, UTF-16, and UTF-8. `ConvertUTF.h` is created by Unicode, Inc., so the UTF conversion API is not explicitly defined in this section. For more information, refer to `ConvertUTF.h`.

`mpeos_context.h` provides the interface for defining and managing a graphics context within which rendering takes place. The graphics context is defined by the following attributes: surface, color, font, origin, clipping rectangle, and paint mode.

`mpeos_draw.h` provides drawing and block copy operations on graphics surfaces.

`mpeos_font.h` provides the interface for creating a font and providing various matrix information on a given font.

`mpeos_fontfact.h` provides the interface to link fonts to an associated font factory.

`mpeos_screen.h` provides information on the current display screen. The screen capabilities contain information such as screen resolution, bit depth, etc.

`mpeos_surface.h` provides the interface for defining and managing off-screen graphic surfaces.

`mpeos_uievent.h` monitors the system for user input events.

Graphics reference model

The Graphic API provides functions for the graphics plane. The background and video plane functionality is provided by the Display API. This section provides information about the following topics:

Screen layout

Coordinate system

Screen layout

The screen order from back to front is background, video, graphics. However, there may be more than one object of each type (for example, 1 background, 2 video, and 1 graphics).

Background

The background plane typically displays one background color, but on some platforms an image can be displayed. The background plane is always full-screen and is always composited with the video plane using a source rule. If the video plane is full-screen, the background plane is not visible.

Graphics

The graphics plane is used for on-screen graphics, for example, subtitles, on-screen menus, Emergency Alert System (EAS), and application graphics.

The graphics plane is typically limited in the number of colors or in the size of the plane. The minimum support for color formats is 2-bit alpha and 24-bit RGB. ARGB8888 is recommended. Other formats, such as RGB888 and RGB565 may be supported to save space for off-screen images, but are insufficient for the main screen surface.

-
- ◆ **NOTE:** The OCAP stack supports both 640x480 and 960x540 graphic-device resolution for standard- and high-definition.
-

Video

The video plane is positioned in front of the background plane. Scaling and position of the video plane is limited based on the target environment. Scaling is typically limited to full-screen or quarter-screen. Positioning of the video plane is typically limited to certain areas on the screen.

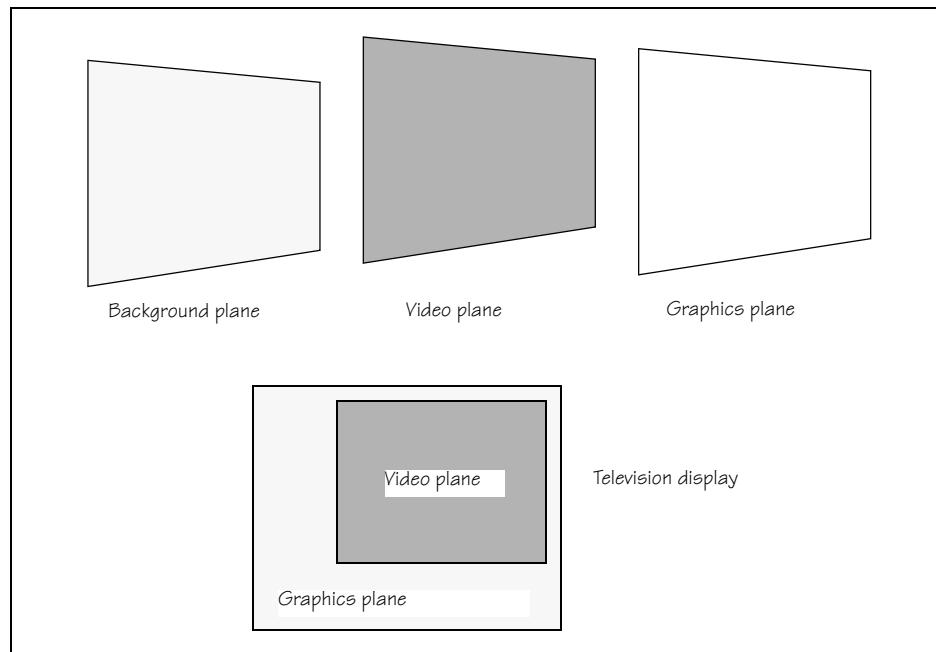


FIGURE GFX-1:
Planes and television display

Coordinate system

The graphics coordinate system is anchored in the upper left-hand corner of the screen, with coordinates increasing down and to the right. Coordinates lie between pixels of the screen. Operations that draw outlines of shapes traverse a coordinate path with a pen that hangs beneath and to the right of the path. The size of the pen is always one pixel in width and height.

◆ **NOTE:** Information in this section is partly taken from [Graphic Java: Mastering the JFC Volume I: AWT](#).

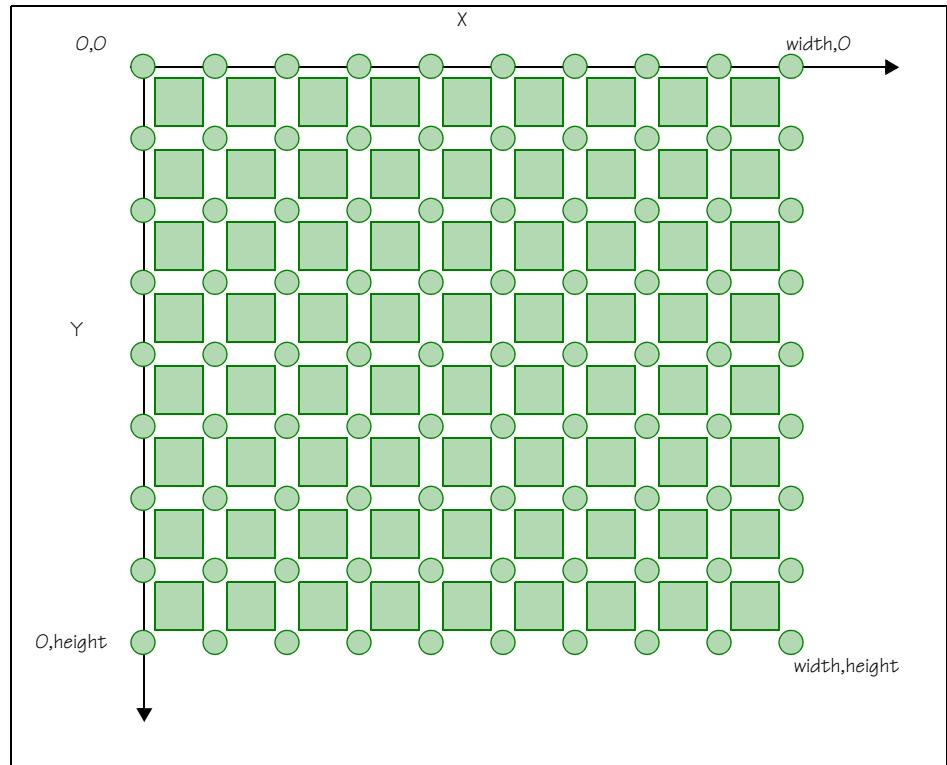


FIGURE GFX-2:

Graphics coordinate system

It is perfectly valid to refer to a location using a negative coordinate. Negative coordinates indicate an imaginary point off of the screen (for screen coordinates) or a point relative to the translated origin. In some cases (for example, `mpeos_gfxBitBlt` and `mpeos_gfxStretchBlt`), it is also valid to use a negative width or height to indicate a translation about one or both axis.

Surface

A surface corresponds to the graphics plane of the physical screen or an off-screen pixel map. Rendering operations are performed on a drawing surface through a specific context. Synonyms include bitmap, pixel map (or pixmap), or raster.

Graphic context

A graphic context provides the context through which rendering operations take place. A graphic context defines and manages the attributes that affect rendering.

Fonts

Fonts define the glyphs used to render text on the screen. OCAP requires support of PFR fonts but the actual support for specific font formats is within the MPEOS implementation for a given platform.

General definitions

The following definitions, which are defined in `mpe_uievents.h`, `mpeos_gfx.h`, and `mpeos_screen.h`, are used by the graphics functions:

| | |
|---------------------------------|------------------------------------|
| <code>GFX_LOCK</code> | <code>GFX_UNLOCK</code> |
| <code>MPE_GFX_UNKNOWN</code> | <code>MPE_GFX_WAIT_INFINITE</code> |
| <code>mpe_gfxArgbToColor</code> | <code>mpe_gfxGetAlpha</code> |
| <code>mpe_gfxGetBlue</code> | <code>mpe_gfxGetGreen</code> |
| <code>mpe_gfxGetRed</code> | <code>mpe_gfxRgbToColor</code> |
| <code>OCAF_KEY_PRESS</code> | <code>OCAF_KEY_RELEASE</code> |

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the defined name as the values may change in a future release.

`GFX_LOCK`

`GFX_LOCK` specifies a lock on the surface. `GFX_LOCK` is defined in `mpeos_screen.h` as follows:

```
#define GFX_LOCK() mpeos_mutexAcquire(_screen.surf->mutex)
```

`GFX_UNLOCK`

`GFX_UNLOCK` specifies an unlock on the surface. `GFX_UNLOCK` is defined in `mpeos_screen.h` as follows:

```
#define GFX_UNLOCK() mpeos_mutexRelease(_screen.surf->mutex)
```

`MPE_GFX_UNKNOWN`

`MPE_GFX_UNKNOWN` defines an unknown value as 0. `MPE_GFX_UNKNOWN` is defined in `mpeos_gfx.h` as follows:

```
#define MPE_GFX_UNKNOWN (0)
```

`MPE_GFX_WAIT_INFINITE`

`MPE_GFX_WAIT_INFINITE` infinitely waits for `gfxWaitNextEvent()` to fill with the event data. `MPE_GFX_WAIT_INFINITE` is defined in `mpeos_gfx.h` as follows:

```
#define MPE_GFX_WAIT_INFINITE 0xFFFFFFFF
```

`mpe_gfxArgbToColor`

`mpe_gfxArgbToColor` creates a color value from Alpha, Red, Green, and Blue (ARGB) values. `mpe_gfxArgbToColor` is defined in `mpeos_gfx.h` as follows:

```
#define mpe_gfxArgbToColor(a,r,g,b)
    ((mpe_GfxColor)((((a)&0xFF) << 24) | (((r)&0xFF) << 16) |
     ((g)&0xFF) << 8) | ((b)&0xFF)))
```

`mpe_gfxGetAlpha`

`mpe_gfxGetAlpha` extracts the alpha value from an `mpe_GfxColor` enumeration. `mpe_gfxGetAlpha` is defined in `mpeos_gfx.h` as follows:

```
#define mpe_gfxGetAlpha(argb) ((uint8_t)((argb) >> 24)
    & 0xff))
```

mpe_gfxGetBlue

mpe_gfxGetBlue extracts the blue value from an mpe_GfxColor enumeration. mpe_gfxGetBlue is defined in mpeos_gfx.h as follows:

```
#define mpe_gfxGetBlue(argb) ((uint8_t)((argb) & 0xff))
```

mpe_gfxGetGreen

mpe_gfxGetGreen extracts the green value from an mpe_GfxColor enumeration. mpe_gfxGetGreen is defined in mpeos_gfx.h as follows:

```
#define mpe_gfxGetGreen(argb) (((uint8_t)((argb) >> 8) & 0xff))
```

mpe_gfxGetRed

mpe_gfxGetRed extracts the red value from an mpe_GfxColor enumeration. mpe_gfxGetRed is defined in mpeos_gfx.h as follows:

```
#define mpe_gfxGetRed(argb) (((uint8_t)((argb) >> 16) & 0xff))
```

mpe_gfxRgbToColor

mpe_gfxRgbToColor creates a color value from Red, Green, and Blue (RGB) values. mpe_gfxRgbToColor is defined in mpeos_gfx.h as follows:

```
#define mpe_gfxRgbToColor(r,g,b) mpe_gfxArgbToColor  
(0xFF, r, g, b)
```

OCAP_KEY_PRESS

OCAP_KEY_PRESS specifies a key down event. Events are generated when the key is pressed. OCAP_KEY_PRESS is defined in mpe_uievents.h as follows:

```
#define OCAP_KEY_PRESSED 401L
```

OCAP_KEY_RELEASE

OCAP_KEY_RELEASE specifies a key up event. Events are generated when the key is released. OCAP_KEY_RELEASE is defined in mpe_uievents.h as follows:

```
#define OCAP_KEY_RELEASED 402L
```

Supported data types and structures

The graphics-manager API contains the following data types and structures:

- ◆ *General data types and structures*
- ◆ *Context data types and structures*
- ◆ *Font data types and structures*
- ◆ *Font factory data types and structures*
- ◆ *Screen data types and structures*
- ◆ *Surface data types and structures*

General data types and structures

The following general data types and structures, which are defined in `mpe_error.h`, `mpeos_gfx.h`, and `mpe_types.h`, are used by the graphic functions:

| | |
|----------------------------------|---------------------------------|
| <code>mpe_Bool</code> | <code>mpe_Error</code> |
| <code>mpe_GfxAlphaChannel</code> | <code>mpe_GfxBitDepth</code> |
| <code>mpe_GfxColor</code> | <code>mpe_GfxColorFormat</code> |
| <code>mpe_GfxContext</code> | <code>mpe_GfxDimensions</code> |
| <code>mpe_GfxError</code> | <code>mpe_GfxEvent</code> |
| <code>mpe_GfxFont</code> | <code>mpe_GfxFontDesc</code> |
| <code>mpe_GfxFontFactory</code> | <code>mpe_GfxFontFormat</code> |
| <code>mpe_GfxFontMetrics</code> | <code>mpe_GfxFontStyle</code> |
| <code>mpe_GfxPaintMode</code> | <code>mpe_GfxPalette</code> |
| <code>mpe_GfxPoint</code> | <code>mpe_GfxRectangle</code> |
| <code>mpe_GfxScreen</code> | <code>mpe_GfxSurface</code> |
| <code>mpe_GfxSurfaceInfo</code> | <code>mpe_GfxWchar</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The Multimedia Platform Environment (MPE) error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

mpe_GfxAlphaChannel

`mpe_GfxAlphaChannel` defines information about the alpha channel.
`mpe_GfxAlphaChannel` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxAlphaChannel {
    mpe_GfxColorFormat format;
    mpe_GfxBitDepth bpp;
    void *data;
    uint32_t widthbytes;
} mpe_GfxAlphaChannel;
```

where

| | |
|--------------------------------|--|
| <code>format</code> | specifies the color format for the alpha data. <code>mpe_GfxColorFormat</code> is described in the <i>mpe_GfxColorFormat</i> section. Currently, the color format may be one of the following values: |
| <code>MPE_GFX_RGB888</code> | specifies 24-bits per pixel with no alpha channel. |
| <code>MPE_GFX_RGB565</code> | specifies 16-bits per pixel with no alpha channel. |
| <code>MPE_GFX_ARGB8888</code> | specifies 32-bits per pixel and a separate 8-bits per pixel alpha channel. |
| <code>MPE_GFX_ARGB1555</code> | specifies 16-bits per pixel and a separate 1-bit per pixel alpha channel. |
| <code>MPE_GFX_UNDEFINED</code> | specifies that the format is undefined. |
| <code>MPE_CLUT8</code> | specifies 8-bits per pixel color lookup index. |
| <code>MPE_GFX_A8RGB888</code> | specifies a 24-bit RGB color with separate 8-bit alpha. |
| <code>MPE_GFX_A2RGB565</code> | specifies a 16-bit RGB color with separate 2-bit alpha. |
| <code>bpp</code> | specifies the number of bits per pixel for the alpha data. <code>mpe_GfxBitDepth</code> is described in the <i>mpe_GfxBitDepth</i> section. Currently, the number of bits per pixel may be one of the following values: |
| <code>MPE_GFX_1BPP</code> | specifies the bit depth is 1-bit per pixel. |
| <code>MPE_GFX_2BPP</code> | specifies the bit depth is 2-bit per pixel. |

MPE_GFX_4BPP

specifies the bit depth is 4-bit per pixel.

MPE_GFX_8BPP

specifies the bit depth is 8-bits per pixel.

data is a pointer to the data to use for the alpha channel.

widthbytes specifies the number of bytes per line of alpha data.

mpe_GfxBitDepth

`mpe_GfxBitDepth` specifies the number of bits per pixel. `mpe_GfxBitDepth` is defined in `mpeos_gfx.h` as follows:

```
typedef enum mpe_GfxBitDepth {
    MPE_GFX_1BPP = 1,
    MPE_GFX_2BPP = 2,
    MPE_GFX_4BPP = 4,
    MPE_GFX_8BPP = 8,
    MPE_GFX_16BPP = 16,
    MPE_GFX_24BPP = 24,
    MPE_GFX_32BPP = 32,
    MPE_GFX_BPP_MAX
} mpe_GfxBitDepth;
```

where

`MPE_GFX_1BPP` specifies the bit depth is 1-bit per pixel.

`MPE_GFX_2BPP` specifies the bit depth is 2-bit per pixel.

`MPE_GFX_4BPP` specifies the bit depth is 4-bit per pixel.

`MPE_GFX_8BPP` specifies the bit depth is 8-bits per pixel.

`MPE_GFX_16BPP` specifies the bit depth is 16-bits per pixel.

`MPE_GFX_24BPP` specifies the bit depth is 24-bits per pixel.

`MPE_GFX_32BPP` specifies the bit depth is 32-bits per pixel.

`MPE_GFX_BPP_MAX`

specifies the maximum value that can be stored in `mpe_GfxBitDepth`.

mpe_GfxColor

`mpe_GfxColor` specifies a 32-bit color value representing alpha, red, green, and blue quadruplet. `mpe_GfxColor` is defined in `mpeos_gfx.h` as follows:

```
typedef uint32_t mpe_GfxColor;
```

mpe_GfxColorFormat mpe_GfxColorFormat defines the color formats. mpe_GfxColorFormat is defined in mpeos_gfx.h as follows:

```
typedef enum mpe_GfxColorFormat {
    MPE_GFX_RGB888,
    MPE_GFX_RGB565,
    MPE_GFX_ARGB8888,
    MPE_GFX_ARGB1555,
    MPE_GFX_UNDEFINED,
    MPE_GFX_CLUT8,
    MPE_GFX_A8RGB888,
    MPE_GFX_A2RGB565,
    MPE_GFX_COLOR_FORMAT_MAX
} mpe_GfxColorFormat;
```

where

MPE_GFX_RGB888

specifies 24-bits per pixel with no alpha channel.

MPE_GFX_RGB565

specifies 16-bits per pixel with no alpha channel.

MPE_GFX_ARGB8888

specifies 32-bits per pixel and a separate 8-bits per pixel alpha channel.

MPE_GFX_ARGB1555

specifies 16-bits per pixel and a separate 1-bit per pixel alpha channel.

MPE_GFX_UNDEFINED

specifies that the format is undefined.

MPE_CLUT8

specifies 8-bits per pixel color lookup index.

MPE_GFX_A8RGB888

specifies a 24-bit RGB color with separate 8-bit alpha.

MPE_GFX_A2RGB565

specifies a 16-bit RGB color with separate 2-bit alpha.

MPE_GFX_COLOR_FORMAT_MAX

specifies the maximum value that can be stored in mpe_GfxColorFormat.

mpe_GfxContext

`mpe_GfxContext` is a handle to a graphics context. `mpe_GfxContext` is defined in `mpeos_gfx.h` as follows:

```
typedef struct _mpeH_GfxContext_t {int unused1;}  
*mpe_GfxContext;
```

mpe_GfxDimensions

`mpe_GfxDimensions` specifies the dimensions of the graphics element. `mpe_GfxDimensions` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxDimensions {  
    int32 width;  
    int32 height;  
} mpe_GfxDimensions;
```

where

| | |
|---------------------|--|
| <code>width</code> | indicates the width in pixels of the graphic element. |
| <code>height</code> | indicates the height in pixels of the graphic element. |

mpe_GfxError

`mpe_GfxError` defines the error codes returned by the graphics functions. `mpe_GfxError` is defined in `mpeos_gfx.h` as follows:

```
typedef enum mpe_GfxError {  
    MPE_GFX_ERROR_NOERR,  
    MPE_GFX_ERROR_UNKNOWN,  
    MPE_GFX_ERROR_OSERR,  
    MPE_GFX_ERROR_NOMEM,  
    MPE_GFX_ERROR_INVALID,  
    MPE_GFX_ERROR_NOT_SUPPORTED,  
    MPE_GFX_ERROR_OUT_OF_BOUNDS,  
    MPE_GFX_ERROR_UNIMPLEMENTED,  
    MPE_GFX_ERROR_FALSE,  
    MPE_GFX_ERROR_FF_DEL_PENDING,  
    MPE_GFX_ERROR_NOFONT,  
    MPE_GFX_ERROR_FONTFORMAT  
} mpe_GfxError;
```

where

`MPE_GFX_ERROR_NOERR`
indicates no error.

`MPE_GFX_ERROR_UNKNOWN`
indicates a generic error.

`MPE_GFX_ERROR_OSERR`
indicates the operating system failed.

`MPE_GFX_ERROR_NOMEM`
indicates there is not enough memory available to perform the specified graphics operation.

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter was passed into a function.

`MPE_GFX_ERROR_NOT_SUPPORTED`
indicates the feature is not supported.

MPE_GFX_ERROR_OUT_OF_BOUNDS
 indicates the graphic area is out of bounds.

MPE_GFX_ERROR_UNIMPLEMENTED
 indicates the requested operation is not implemented.

MPE_GFX_ERROR_FALSE
 indicates the statement is false.

MPE_GFX_ERROR_FF_DEL_PENDING
 indicates the font factory deletion is pending.

MPE_GFX_ERROR_NOFONT
 indicates no fonts were in the font factory.

MPE_GFX_ERROR_FONTFORMAT
 indicates the font could not be created as it is an invalid format.

mpe_GfxEvent

`mpe_GfxEvent` specifies a user-interface event. `mpe_GfxEvent` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxEvent {
    int32 eventID;
    int32 eventCode;
    int32 eventChar;
    int32 rsvd[1];
} mpe_GfxEvent;
```

where

`eventID` identifies the specific event type. Currently, the following values are defined for `eventID`:

OCAP_KEY_PRESSED
 indicates a key down event. Events are generated when the key is pressed.

OCAP_KEY_RELEASED
 indicates a key up event. Events are generated when the key is released.

`eventCode` indicates the virtual key code. The full set of virtual key codes is defined in `mpe_uievents.h`. The following virtual key codes examples are defined for `eventCode`:

OCAP_VK_DOWN
 specifies the virtual channel-down key is pressed.

OCAP_VK_UP
 specifies the virtual channel-up key is pressed.

OCAP_VK_0
 specifies the virtual 0 key is pressed.

OCAP_VK_EXIT
 specifies the virtual exit key is pressed.

| | |
|-----------|---|
| eventChar | if known, reports the 16-bit Unicode character for a key event back to the caller of the function. Otherwise, one of the following values are sent to the caller: |
| | OCAP_CHAR_UNDEFINED specifies no Unicode character is defined for this event. |
| | OCAP_CHAR_UNKNOWN specifies the Unicode character is unknown. The AWT implementation makes a best guess based upon the eventCode and the known state of modifier keys. |

-
- ◆ **NOTE:** In order to work with older ports where eventChar is not supported, the Abstract Windowing Toolkit (AWT) implementation sets the value of eventChar to OCAP_KEY_UNKNOWN before calling waitNextUiEvent().

If eventChar==OCAP_KEY_UNKNOWN on return from waitNextUiEvent(), the caller (AWT) generates the eventChar based upon the eventCode and the current state of standard US keyboard modifier keys. However, it is preferred that the MPEOS port, if possible, fill in eventChar as it enables a richer set of eventChar to be generated from a potentially larger set of keyboard types.

rsvd[1] reserved for future use.

mpe_GfxFont

mpe_GfxFont is a handle to a graphics font. mpe_GfxFont is defined in mpeos_gfx.h as follows:

```
typedef struct _mpeH_GfxFont_t {int unused1;} *mpe_GfxFont;
```

mpe_GfxFontDesc

`mpe_GfxFontDesc` provides a description of a font. `mpe_GfxFontDesc` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxFontDesc {
    mpe_GfxWchar *name;
    uint32_t namelength;
    mpe_GfxFontFormat fnt_format;
    mpe_GfxFontStyle style;
    int32 minsize;
    int32 maxsize;
    uint8_t *data;
    uint32_t datasize;
    struct mpe_GfxFontDesc *prev;
    struct mpe_GfxFontDesc *next;
} mpe_GfxFontDesc;
```

where

| | |
|-------------------------|---|
| <code>name</code> | is a pointer to the name of the font. <code>mpe_GfxWchar</code> is described in the <i>mpe_GfxWchar</i> section. |
| <code>namelength</code> | indicates the number of characters in the name. |
| <code>fnt_format</code> | indicates the format of the font. <code>mpe_GfxFontFormat</code> is described in the <i>mpe_GfxFontFormat</i> section. |
| <code>style</code> | specifies which style of font to use. <code>mpe_GfxFontStyle</code> is described in the <i>mpe_GfxFontStyle</i> section. Currently, the font styles may be one of the following values: |
| | <code>MPE_GFX_PLAIN</code> |
| | specifies the font does not have bold or italics formatting. |
| | <code>MPE_GFX_BOLD</code> |
| | specifies the font is bold. |
| | <code>MPE_GFX_ITALIC</code> |
| | specifies the font is italic. |
| | <code>MPE_GFX_BOLD_ITALIC</code> |
| | specifies the font is bold and italic. |
| <code>minsize</code> | indicates the minimum point size that can be specified. |
| <code>maxsize</code> | indicates the maximum point size that can be specified. |
| <code>data</code> | is a pointer to font data in a buffer. |
| <code>datasize</code> | indicates the size in bytes of the buffer pointed to by <code>data</code> . |
| <code>prev</code> | is a pointer to the previous font description in the font factory. <code>mpe_GfxFontDesc</code> is described in the <i>mpe_GfxFontDesc</i> section. |
| <code>next</code> | is a pointer to the next font description in the font factory. <code>mpe_GfxFontDesc</code> is described in the <i>mpe_GfxFontDesc</i> section. |

mpe_GfxFontFactory `mpe_GfxFontFactory` is a handle to a graphics font factory.

`mpe_GfxFontFactory` is defined in `mpeos_gfx.h` as follows:

```
typedef struct _mpeH_GfxFontFactory_t {int unused1;}  
*mpe_GfxFontFactory;
```

mpe_GfxFontFormat `mpe_GfxFontFormat` provides the format of the font. `mpe_GfxFontFormat` is defined in `mpeos_gfx.h` as follows:

```
typedef enum mpe_GfxFontFormat {  
    GFX_FONT_PFR,  
    GFX_FONTFORMAT_MAX  
} mpe_GfxFontFormat;
```

where

`GFX_FONT_PFR`

indicates the font is in a portable font resource format.

`GFX_FONTFORMAT_MAX`

specifies the maximum value that can be stored in `mpe_GfxFontFormat`.

mpe_GfxFontMetrics `mpe_GfxFontMetrics` defines the font matrix. `mpe_GfxFontMetrics` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxFontMetrics {  
    int16 height;  
    int16 ascent;  
    int16 descent;  
    int16 leading;  
    int16 maxadvance;  
    int16 maxascent;  
    int16 maxdescent;  
    int16 first_char;  
    int16 last_char;  
} mpe_GfxFontMetrics;
```

where

`height` indicates the total height in pixels of the font. This is the sum of `ascent` + `descent` + `leading`.

`ascent` indicates the number of pixels in a glyph above the baseline.

`descent` indicates the number of pixels in a glyph below the baseline.

`leading` indicates the interline spacing expected to be reserved between the `descent` and `ascent` of adjacent lines.

`maxadvance` indicates the maximum advance width of any character in this font. `maxadvance` may be -1 if the maximum advance width is not known.

| | |
|-------------------------|--|
| <code>maxascent</code> | indicates the maximum ascent for any code in this font. |
| <code>maxdescent</code> | indicates the maximum descent for any code in this font. |
| <code>first_char</code> | indicates the first character in the font character map. |
| <code>last_char</code> | indicates the last character in the font character map. |

mpe_GfxFontStyle

`mpe_GfxFontStyle` defines the available font styles. `mpe_GfxFontStyle` is defined in `mpeos_gfx.h` as follows:

```
typedef enum mpe_GfxFontStyle {
    MPE_GFX_PLAIN,
    MPE_GFX_BOLD,
    MPE_GFX_ITALIC,
    MPE_GFX_BOLD_ITALIC,
    MPE_GFX_FNT_STYLE_MAX
} mpe_GfxFontStyle;
```

where

`MPE_GFX_PLAIN`
specifies the font does not have bold or italics formatting.

`MPE_GFX_BOLD`
specifies the font is bold.

`MPE_GFX_ITALIC`
specifies the font is italic.

`MPE_GFX_BOLD_ITALIC`
specifies the font is bold and italic.

`MPE_GFX_FNT_STYLE_MAX`
specifies the maximum value that can be stored in
`mpe_GfxFontStyle`.

mpe_GfxPaintMode

`mpe_GfxPaintMode` identifies the modes available for the paint operation. `mpe_GfxPaintMode` is defined in `mpeos_gfx.h` as follows:

```
typedef enum mpe_GfxPaintMode {
    MPE_GFX_XOR = 0,
    MPE_GFX_CLR = 1,
    MPE_GFX_SRC = 2,
    MPE_GFX_SRCOVER = 3,
    MPE_GFX_DSTOVER = 4,
    MPE_GFX_SRCIN = 5,
    MPE_GFX_DSTIN = 6,
    MPE_GFX_SRCOUT = 7,
    MPE_GFX_DSTOUT = 8,
    MPE_GFX_DST = 9,
    MPE_GFX_MODE_MAX
} mpe_GfxPaintMode;
```

where

- MPE_GFX_XOR indicates an exclusive or (XOR) bit operation.
 $Cd=Cd \wedge (Cs \wedge Cx)$; where Cd , Cs , and Cx are respectively destination, source, and XOR color values. Alpha value is ignored.
- MPE_GFX_CLR indicates the following composition rule: $Cd=0$, $Ad=0$; where Ad is the destination alpha value.
- MPE_GFX_SRC indicates the following composition rule:
 $Cd=Cs$, $Ad=As$; where As is the source alpha value.
- MPE_GFX_SRCOVER indicates the following composition rule:
 $Cd=Cs+Cd*(1-As)$ and $Ad=As+Ad*(1-As)$.
- MPE_GFX_DSTOVER indicates the following composition rule:
 $Cd=Cs*(1-Ad)+Cd$ and $Ad=As*(1-Ad)+Ad$.
- MPE_GFX_SRCIN indicates the following composition rule:
 $Cd=Cs*Ad$ and $Ad=As*Ad$.
- MPE_GFX_DSTIN indicates the following composition rule:
 $Cd=Cd*As$ and $Ad=Ad*As$.
- MPE_GFX_SRCOUT indicates the following composition rule:
 $Cd=Cs*(1-Ad)$ and $Ad=As*(1-Ad)$.
- MPE_GFX_DSTOUT indicates the following composition rule:
 $Cd=Cd*(1-As)$ and $Ad=Ad*(1-As)$.
- MPE_GFX_DST indicates the following composition rule:
 $Cd=Cd$ and $Ad=Ad$.
- MPE_GFX_MODE_MAX specifies the maximum value that can be stored in `mpe_GfxPaintMode`.

`mpe_GfxPalette`

`mpe_GfxPalette` is a handle to a color palette. `mpe_GfxPalette` is defined in `mpeos_gfx.h` as follows:

```
typedef struct _mpeH_GfxPalette_t {int unused1;}  
*mpe_GfxPalette;
```

mpe_GfxPoint

`mpe_GfxPoint` defines a point on the screen. `mpe_GfxPoint` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxPoint {
    int32 x;
    int32 y;
} mpe_GfxPoint;
```

where

- x indicates the x coordinate to use for the point.
- y indicates the y coordinate to use for the point.

mpe_GfxRectangle

`mpe_GfxRectangle` defines the origin and dimensions of a rectangle for graphics. `mpe_GfxRectangle` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxRectangle {
    int32 x;
    int32 y;
    int32 width;
    int32 height;
} mpe_GfxRectangle;
```

where

- x indicates the x coordinate to use as the bottom-left corner of the rectangle.
- y indicates the y coordinate to use as the bottom-left corner of the rectangle.
- width indicates the width in pixels of the rectangle.
- height indicates the height in pixels of the rectangle.

mpe_GfxScreen

`mpe_GfxScreen` is a handle to a graphics screen. `mpe_GfxScreen` is defined in `mpeos_gfx.h` as follows:

```
typedef struct _mpeH_GfxScreen_t {int unused1;} *mpe_GfxScreen;
```

mpe_GfxSurface

`mpe_GfxSurface` is a handle to a graphics surface. `mpe_GfxSurface` is defined in `mpeos_gfx.h` as follows:

```
typedef struct _mpeH_GfxSurface_t {int unused1;}
*mpe_GfxSurface;
```

mpe_GfxSurfaceInfo

`mpe_GfxSurfaceInfo` provides information about the surface. `mpe_GfxSurfaceInfo` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxSurfaceInfo {
    mpe_GfxColorFormat format;
    mpe_GfxBitDepth bpp;
    mpe_GfxDimensions dim;
    uint32_t widthbytes;
    void *pixeldata;
    mpe_GfxPalette clut;
} mpe_GfxSurfaceInfo;
```

where

format

specifies the color format for the surface.

`mpe_GfxColorFormat` is described in the *mpe_GfxColorFormat* section. Currently, the color format may be one of the following values:

MPE_GFX_RGB888

specifies 24-bits per pixel with no alpha channel.

MPE_GFX_RGB565

specifies 16-bits per pixel with no alpha channel.

MPE_GFX_ARGB8888

specifies 32-bits per pixel and a separate 8-bits per pixel alpha channel.

MPE_GFX_ARGB1555

specifies 16-bits per pixel and a separate 1-bit per pixel alpha channel.

MPE_GFX_UNDEFINED

specifies that the format is undefined.

MPE_CLUT8

specifies 8-bits per pixel color lookup index.

MPE_GFX_A8RGB888

specifies a 24-bit RGB color with separate 8-bit alpha.

MPE_GFX_A2RGB565

specifies a 16-bit RGB color with separate 2-bit alpha.

bpp

specifies the number of bits per pixel. `mpe_GfxBitDepth` is described in the *mpe_GfxBitDepth* section. Currently, the number of bits per pixel may be one of the following values:

MPE_GFX_1BPP

specifies the bit depth is 1-bit per pixel.

MPE_GFX_2BPP

specifies the bit depth is 2-bit per pixel.

MPE_GFX_4BPP

specifies the bit depth is 4-bit per pixel.

MPE_GFX_8BPP

specifies the bit depth is 8-bits per pixel.

MPE_GFX_16BPP

specifies the bit depth is 16-bits per pixel.

MPE_GFX_24BPP

specifies the bit depth is 24-bits per pixel.

MPE_GFX_32BPP

specifies the bit depth is 32-bits per pixel.

| | |
|-----------------|--|
| dim | specifies the dimensions of the graphics element. <i>mpe_GfxDimensions</i> is described in the <i>mpe_GfxDimensions</i> section. Currently, the following values are defined for the width and height dimensions in pixels: |
| MPE_GFX_640x480 | specifies the graphics resolution is 640 x 480 pixels. |
| MPE_GFX_320x240 | specifies the graphics resolution is 320 x 240 pixels. |
| widthbytes | indicates the bytes per line. |
| pixeldata | is a pointer to the pixel data. |
| clut | indicates the color palette, if the format is MPE_GFX_CLUT8. <i>mpe_GfxPalette</i> is described in the <i>mpe_GfxPalette</i> section. |

mpe_GfxWchar

mpe_GfxWchar specifies a wide character. *mpe_GfxWchar* is defined in *mpeos_gfx.h* as follows:

```
typedef uint16_t mpe_GfxWchar;
```

Context data types and structures

The following context related data types and structures, which are defined in `mpeos_context.h` and `os_gfx.h`, are used by the graphic functions:

| | |
|-------------------------------|----------------------------|
| <code>mpeos_GfxContext</code> | <code>os_GfxContext</code> |
|-------------------------------|----------------------------|

`mpeos_GfxContext`

`mpeos_GfxContext` specifies the graphic context internal representation. `mpeos_GfxContext` is defined in the `mpeos_context.h` as follows:

```
typedef struct mpeos_GfxContext {
    mpe_GfxColor color;
    mpe_GfxFont font;
    mpe_GfxPoint orig;
    mpe_GfxRectangle cliprect;
    mpe_GfxPaintMode paintmode;
    uint32_t modeData;
    mpeos_GfxSurface *surf;
    os_GfxContext os_ctx;
} mpeos_GfxContext;
```

where

| | |
|------------------------|--|
| <code>color</code> | specifies the current drawing color. <code>mpe_GfxColor</code> is described in the <i>mpe_GfxColor</i> section. |
| <code>font</code> | specifies the handle to the current font. <code>mpe_GfxFont</code> is described in the <i>mpe_GfxFont</i> section. |
| <code>orig</code> | specifies the origin from drawing and clipping area. <code>mpe_GfxPoint</code> is described in the <i>mpe_GfxPoint</i> section. |
| <code>cliprect</code> | specifies the current clipping rectangle. <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |
| <code>paintmode</code> | specifies the current paint mode. <code>mpe_GfxPaintMode</code> is described in the <i>mpe_GfxPaintMode</i> section. |
| <code>modeData</code> | specifies the data associated with the <code>paintmode</code> . For XOR this is a color and for Porter-Duff it is a constant alpha value. |
| <code>surf</code> | is a pointer to a graphics surface. <code>mpe_GfxSurface</code> is described in the <i>mpe_GfxSurface</i> section. |
| <code>os_ctx</code> | specifies the operating system-specific definition of the graphic context. <code>os_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |

`os_GfxContext`

`os_GfxContext` specifies the operating system-specific definition of the graphic context. The contents of this data structure is completely defined by the port.

Font data types and structures

The following context related structure, which is defined in `mpeos_font.h`, is used by the graphic functions:

| | |
|----------------------------|-------------------------|
| <code>mpeos_GfxFont</code> | <code>os_GfxFont</code> |
| <code>os_Mutex</code> | |

`mpeos_GfxFont`

`mpeos_GfxFont` specifies the font internal representation. `mpeos_GfxFont` is defined as follows:

```
typedef struct mpeos_GfxFont {
    os_GfxFont osf;
    mpe_GfxFontFactory ff;
    mpe_GfxWchar *name;
    uint32_t namelength;
    uint32_t size;
    mpe_GfxFontStyle style;
    uint32_t refCount;
    os_Mutex mutex;
    struct mpeos_GfxFont *prev;
    struct mpeos_GfxFont *next;
} mpeos_GfxFont;
```

where

| | |
|-------------------------|---|
| <code>osf</code> | specifies the operating system-specific font definition. <code>os_GfxFont</code> is described in the <i>os_GfxFont</i> section. |
| <code>ff</code> | specifies the font factory from which it was created. <code>mpe_GfxFontFactory</code> is described in the <i>mpe_GfxFontFactory</i> section. |
| <code>name</code> | is a pointer to the family font name. <code>mpe_GfxWchar</code> is described in the <i>mpe_GfxWchar</i> section. |
| <code>namelength</code> | specifies the number of wide characters in the string. |
| <code>size</code> | specifies the point size of the font. |
| <code>style</code> | specifies the font style. <code>mpe_GfxFontStyle</code> is described in the <i>mpe_GfxFontStyle</i> section. |
| <code>refCount</code> | specifies the font usage count. |
| <code>mutex</code> | provides thread safe access to <code>refCount</code> . <code>os_Mutex</code> is described in the <i>os_Mutex</i> section. |
| <code>prev</code> | is a pointer to the previous font. |
| <code>next</code> | is a pointer to the next font. |

os_GfxFont

`os_GfxFont` specifies the font internal representation. `os_GfxFont` is defined as follows:

```
typedef struct os_GfxFont {  
    LOGFONT lf;  
    HFONT hFont;  
    TEXTMETRIC tm;  
} os_GfxFont;
```

where

`lf` specifies the logical font descriptor.

`hFont` specifies the Windows font handle.

`tm` specifies the text metrics information.

os_Mutex

`os_Mutex` specifies the font internal representation. `os_Mutex` is defined as follows:

```
typedef struct _os_Mutex_Struct *os_Mutex;
```

Font factory data types and structures

The following font factory related data types and structures, which are defined in `mpeos_fontfact.h`, are used by the graphic functions:

`mpeos_GfxFontFactory`

`mpeos_GfxFontFactory`

`mpeos_GfxFontFactory` specifies the font factory internal representation. `mpeos_GfxFontFactory` is defined as follows:

```
typedef struct mpeos_GfxFontFactory {  
    os_Mutex mutex;  
    mpe_Bool tbd;  
    uint32_t fontCount;  
    mpe_GfxFontDesc *descList[2];  
    mpeos_GfxFont *fontList[2];  
} mpeos_GfxFontFactory;
```

where

`mutex` provides thread safe access to `descList` and `fontList`. `os_Mutex` is described in the `os_Mutex` section.

`tbd` specifies whether the representation is deleted. If `tbd` is TRUE, the representation is deleted. If `tbd` is FALSE, the representation is not deleted. `mpe_Bool` is described in the `mpe_Bool` section.

`fontCount` specifies the number of active fonts in the factory.

`descList` is a pointer to the list of font descriptions under control of `mpeos_GfxFontFactory`. `mpe_GfxFontDesc` is defined in the `mpe_GfxFontDesc` section.

`fontList` is a pointer to the list of internal font representations under control of `mpeos_GfxFontFactory`. `mpe_GfxFont` is defined in the `mpe_GfxFont` section.

Screen data types and structures

The following screen related data types and structures, which are defined in `mpeos_screen.h` and `os_gfx.h`, are used by the graphic functions:

| | |
|------------------------------|---------------------------|
| <code>mpeos_GfxScreen</code> | <code>os_GfxScreen</code> |
|------------------------------|---------------------------|

`mpeos_GfxScreen`

`mpeos_GfxScreen` specifies the graphic screen internal representation. `mpeos_GfxScreen` is defined as follows:

```
typedef struct mpeos_GfxScreen {
    int32 x;
    int32 y;
    int32 width;
    int32 height;
    int32 widthBytes;
    mpe_GfxColorFormat colorFormat;
    mpe_GfxBitDepth bitDepth;
    mpeos_GfxSurface *surface;
    os_GfxScreen osScr;
} mpeos_GfxScreen;
```

where

- `x` indicates the x coordinate of the screen area.
- `y` indicates the y coordinate of the screen area.
- `width` indicates the width in pixels of the screen.
- `height` indicates the height in pixels of the screen.
- `widthBytes` indicates the bytes per line.
- `colorFormat` defines the color formats. `mpe_GfxColorFormat` is defined in the *mpe_GfxColorFormat* section. Currently, the color format may be one of the following values:

`MPE_GFX_RGB888`

specifies 24-bits per pixel with no alpha channel.

`MPE_GFX_RGB565`

specifies 16-bits per pixel with no alpha channel.

`MPE_GFX_ARGB8888`

specifies 32-bits per pixel and a separate 8-bits per pixel alpha channel.

`MPE_GFX_ARGB1555`

specifies 16-bits per pixel and a separate 1-bit per pixel alpha channel.

| | |
|---------------------|---|
| | MPE_GFX_UNDEFINED specifies that the format is undefined. |
| | MPE_CLUT8 specifies 8-bits per pixel color lookup index. |
| | MPE_GFX_A8RGB888 specifies a 24-bit RGB color with separate 8-bit alpha. |
| | MPE_GFX_A2RGB565 specifies a 16-bit RGB color with separate 2-bit alpha. |
| bitDepth | specifies the number of bits per pixel. <code>mpe_GfxBitDepth</code> is defined in the <i>mpe_GfxBitDepth</i> section. Currently, the bit depth may be one of the following values: |
| | MPE_GFX_1BPP specifies the bit depth is 1-bit per pixel. |
| | MPE_GFX_2BPP specifies the bit depth is 2-bit per pixel. |
| | MPE_GFX_4BPP specifies the bit depth is 4-bit per pixel. |
| | MPE_GFX_8BPP specifies the bit depth is 8-bits per pixel. |
| | MPE_GFX_16BPP specifies the bit depth is 16-bits per pixel. |
| | MPE_GFX_24BPP specifies the bit depth is 24-bits per pixel. |
| | MPE_GFX_32BPP specifies the bit depth is 32-bits per pixel. |
| surface | is a pointer to the graphic screen internal representation. <code>mpeos_GfxSurface</code> is defined in the <i>mpeos_GfxSurface</i> section. |
| osScr | specifies the operating system data. <code>os_GfxScreen</code> is defined in the <i>os_GfxScreen</i> section. |
| os_GfxScreen | <code>os_GfxScreen</code> specifies the operating system-specific definition of the screen context. The contents of this data structure is completely defined by the port. |

Surface data types and structures

The following surface related structures, which are defined in `mpeos_surface.h`, are used by the graphic functions:

| | |
|-------------------------------|----------------------------|
| <code>mpeos_GfxSurface</code> | <code>os_GfxSurface</code> |
|-------------------------------|----------------------------|

`mpeos_GfxSurface`

`mpeos_GfxSurface` specifies the graphic screen internal representation. `mpeos_GfxSurface` is defined as follows:

```
typedef struct mpeos_GfxSurface {
    os_Mutex mutex;
    int32 width;
    int32 height;
    int32 bpl;
    mpe_GfxBitDepth bpp;
    mpe_GfxColorFormat colorFormat;
    void* pixel_data;
    mpe_GfxBitDepth alpha_bpp;
    uint32_t alpha_bpl;
    uint8_t* alpha_data;
    mpe_Bool primary;
    mpe_GfxPalette clut;
    os_GfxSurface os_data;
} mpeos_GfxSurface;
```

where

`mutex` specifies the surface is thread safe. `os_Mutex` is defined in the `os_Mutex` section.

`width` specifies the width of surface in pixels.

`height` specifies the height of surface in pixels.

`bpl` specifies the bytes per line.

`bpp` specifies the bit depth (bits per pixel). `mpe_GfxBitDepth` is defined in the `mpe_GfxBitDepth` section. Currently, the bit depth may be one of the following values:

`MPE_GFX_1BPP`

specifies the bit depth is 1-bit per pixel.

`MPE_GFX_2BPP`

specifies the bit depth is 2-bit per pixel.

`MPE_GFX_4BPP`

specifies the bit depth is 4-bit per pixel.

`MPE_GFX_8BPP`

specifies the bit depth is 8-bits per pixel.

MPE_GFX_16BPP

specifies the bit depth is 16-bits per pixel.

MPE_GFX_24BPP

specifies the bit depth is 24-bits per pixel.

MPE_GFX_32BPP

specifies the bit depth is 32-bits per pixel.

colorFormat specifies the color format. `mpe_GfxColorFormat` is defined in the *mpe_GfxColorFormat* section. Currently, the color format may be one of the following values:

MPE_GFX_RGB888

specifies 24-bits per pixel with no alpha channel.

MPE_GFX_RGB565

specifies 16-bits per pixel with no alpha channel.

MPE_GFX_ARGB888

specifies 32-bits per pixel and a separate 8-bits per pixel alpha channel.

MPE_GFX_ARGB1555

specifies 16-bits per pixel and a separate 1-bit per pixel alpha channel.

MPE_GFX_UNDEFINED

specifies that the format is undefined.

MPE_CLUT8

specifies 8-bits per pixel color lookup index.

MPE_GFX_A8RGB888

specifies a 24-bit RGB color with separate 8-bit alpha.

MPE_GFX_A2RGB565

specifies a 16-bit RGB color with separate 2-bit alpha.

pixel_data is a pointer to the pixel data.

alpha_bp1 specifies the alpha channel bytes per line.

alpha_bpp specifies the alpha channel bit depth. `mpe_GfxBitDepth` is defined in the *mpe_GfxBitDepth* section. Currently, the bit depth may be one of the following values:

MPE_GFX_1BPP

specifies the bit depth is 1-bit per pixel.

MPE_GFX_2BPP

specifies the bit depth is 2-bit per pixel.

MPE_GFX_4BPP

specifies the bit depth is 4-bit per pixel.

MPE_GFX_8BPP

specifies the bit depth is 8-bits per pixel.

MPE_GFX_16BPP

specifies the bit depth is 16-bits per pixel.

MPE_GFX_24BPP

specifies the bit depth is 24-bits per pixel.

MPE_GFX_32BPP

specifies the bit depth is 32-bits per pixel.

alpha_data is a pointer to the alpha channel data.

primary determines if it is an on-screen surface. *mpe_Bool* is defined in the *mpe_Bool* section.

| | |
|---------|--|
| clut | specifies the color palette used (if <code>colorFormat == MPE_GFX_CLUT8</code>). <code>mpe_GfxPalette</code> is defined in the <i>mpe_GfxPalette</i> section. |
| os_data | specifies the operating system-specific surface information. <code>os_GfxSurface</code> is defined in the <i>os_GfxSurface</i> section. |

os_GfxSurface

`os_GfxSurface` specifies the operating system-specific definition of the surface context. The contents of this data structure is completely defined by the port. Example syntax:

```
typedef struct os_GfxSurface {IDirectFBSurface *os_s;}  
os_GfxSurface;
```

Supported functions

The graphics-manager API should contain the following types of functions:

- ◆ *Context functions*
- ◆ *Drawing functions*
- ◆ *Font functions*
- ◆ *Font-factory functions*
- ◆ *Screen functions*
- ◆ *Surface functions*
- ◆ *User-interface functions*

Context functions

The context functions implement the interface for defining and managing a graphics context within which rendering takes place. The graphics context is defined by the following attributes: surface, color, font, origin, clipping rectangle, and paint mode. A context is always attached to a given drawing surface and is the destination of all rendering operations. The surface a context maps to cannot be changed. New contexts can be created for a given drawing surface or from an existing context. The following context functions, which are defined in `mpeos_context.h`, need to be supported:

- `mpeos_gfxContextCreate`
creates a new display context based on an existing one.
- `mpeos_gfxContextDelete`
deallocates the specified context.
- `mpeos_gfxContextNew`
creates a new graphics context.
- `mpeos_gfxGetClipRect`
gets the display context's current clipping rectangle.
- `mpeos_gfxGetColor`
gets the current color used for display rendering operations.
- `mpeos_gfxGetFont`
gets the current font used for text rendering operations.
- `mpeos_gfxGetOrigin`
gets the display context's current translation origin.
- `mpeos_gfxGetPaintMode`
gets the current paint mode.
- `mpeos_gfxGetSurface`
gets the surface to which the display is attached.
- `mpeos_gfxSetClipRect`
sets the display context's current clipping rectangle.

`mpeos_gfxSetColor`
sets the current color used for display rendering operations.

`mpeos_gfxSetFont`
sets the current font used for text rendering operations..

`mpeos_gfxSetOrigin`
sets the display context's current translation origin.

`mpeos_gfxSetPaintMode`
sets the current paint mode.

mpeos_gfxContextCreate

creates a new display context based on an existing one.

syntax `mpe_Error mpeos_gfxContextCreate(
 mpe_GfxContext base,
 mpe_GfxContext * context);`

parameter(s) `base` specifies the handle to the base context to duplicate for the new context. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`context` is an output pointer to the location where the handle to the newly created context is stored. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

value returned If the call is successful, `mpeos_gfxContextCreate()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID` indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxContextCreate()` should create a new context as a duplicate of an existing context. `base` specifies the existing context.

To create the new context, `mpeos_gfxContextCreate()` should copy all the attributes from the existing context, including the following:

- ◆ the surface to which the context is attached
- ◆ the color to use for the context
- ◆ the font to use for the context
- ◆ the origin of the context
- ◆ the clipping rectangle

Once the new context, which is specified by `context`, has been created, modifications to `context` do not affect the context specified by `base`.

Be aware the operations performed using the two contexts affect the same drawing surface.

related function(s) `mpeos_gfxContextDelete`
`mpeos_gfxContextNew`

mpeos_gfxContextDelete

deallocates the specified context.

syntax mpe_Error mpeos_gfxContextDelete(mpe_GfxContext context);

parameter(s) *context* specifies the handle to the display context to deallocate.
context is returned by a previous call to
mpeos_gfxContextNew() or mpeos_gfxContextCreate().
mpe_GfxContext is described in the *mpe_GfxContext* section.

value returned If the call is successful, mpeos_gfxContextDelete() should return
MPE_GFX_ERROR_NOERR. Otherwise, it should return the following error
code:

MPE_GFX_ERROR_INVALID

indicates an invalid parameter (for example, out-of-range
values, null pointers, unknown values, etc.).

description mpeos_gfxContextDelete() should deallocate the given context. If a font is
used by the context the font count is updated and deleted if necessary.

related function(s) mpeos_gfxContextCreate
mpeos_gfxContextNew

mpeos_gfxContextNew

creates a new graphics context.

syntax `mpe_Error mpeos_gfxContextNew(
 mpe_GfxSurface surface,
 mpe_GfxContext * context);`

parameter(s) *surface* specifies the handle to the surface for which a new graphics context should be created. `mpe_GfxSurface` is described in the *mpe_GfxSurface* section.

context is an output pointer to the location where the handle to the newly created context is stored. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

value returned If the call is successful, `mpeos_gfxContextNew()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID` indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxContextNew()` should create a new graphics context. The surface associated with the display context is fixed and cannot be changed.

`mpeos_gfxContextNew()` should initialize the display context attributes in the `mpe_GfxContext` structure as follows:

| | |
|------------------|---|
| <i>color</i> | should be initialized to black, which is fully opaque. Therefore, the following values are initialized: R=0, G=0, B=0, A=255. |
| <i>font</i> | should be initialized to the default system font. |
| <i>orig</i> | should be initialized to (0, 0). No translation is applied. |
| <i>clirect</i> | should be initialized to (0, 0, width, height). No clipping is performed. |
| <i>paintmode</i> | should be initialized to <code>MPE_GFX_SRCOVER</code> . |

related function(s) `mpeos_gfxContextCreate`
`mpeos_gfxContextDelete`

mpeos_gfxGetClipRect

gets the display context's current clipping rectangle.

syntax `mpe_Error mpeos_gfxGetClipRect(
 mpe_GfxContext context,
 mpe_GfxRectangle * rect);`

| | | |
|---------------------|----------------------|--|
| parameter(s) | <code>context</code> | specifies the handle to the context for which to get the clipping rectangle. <code>context</code> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>rect</code> | is an output pointer to the location where the current clipping rectangle is stored. <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |

value returned If the call is successful, `mpeos_gfxGetClipRect()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxGetClipRect()` should get the current clipping rectangle that applies to all rendering operations. The clipping rectangle is applied to the context's drawable surface, so the (x, y) coordinates are relative to the drawing surface (0, 0).

The clipping rectangle for a context is defined as the intersection of the specified rectangle and the bounds of the drawable surface. It is legal for the clipping rectangle to extend beyond the defined surface—for example, you could have a negative x or negative y value, or a width or height that extends beyond the width and height of the surface. However, internally, this should be minimized to the intersection.

The origin translation does not apply to the set/get clipping rectangle.

related function(s) `mpeos_gfxSetClipRect`

mpeos_gfxGetColor

gets the current color used for display rendering operations.

syntax `mpe_Error mpeos_gfxGetColor(mpe_GfxContext context, mpe_GfxColor * color);`

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>context</i> | specifies the handle to the context for which to get the color attributes. <i>context</i> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <i>color</i> | is an output pointer to the location where the current color value is stored. <code>mpe_GfxColor</code> is described in the <i>mpe_GfxColor</i> section. |

value returned If the call is successful, `mpeos_gfxGetColor()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxGetColor()` should get the current color attribute of the display context specified by *context*.

related function(s) `mpeos_gfxSetColor`

mpeos_gfxGetFont

gets the current font used for text rendering operations.

syntax `mpe_Error mpeos_gfxGetFont(
 mpe_GfxContext context,
 mpe_GfxFont * font);`

| | | |
|---------------------|----------------------|--|
| parameter(s) | <code>context</code> | specifies the handle to the context for which to get the current font. <code>context</code> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>font</code> | is an output pointer to the location where the current font is stored. <code>mpe_GfxFont</code> is described in the <i>mpe_GfxFont</i> section. |

value returned If the call is successful, `mpeos_gfxGetFont()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxGetFont()` should get the current font attribute of the display context specified by `context`.

related function(s) `mpeos_gfxSetFont`

mpeos_gfxGetOrigin

gets the display context's current translation origin.

syntax `mpe_Error mpeos_gfxGetOrigin(
 mpe_GfxContext context,
 mpe_GfxPoint * point);`

parameter(s) *context* specifies the handle to the context for which to get the translation origin. *context* is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

point is an output pointer to the location where the current translation origin is stored. `mpe_GfxPoint` is described in the *mpe_GfxPoint* section.

value returned If the call is successful, `mpeos_gfxGetOrigin()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxGetOrigin()` should get the current translation origin used during rendering operations. The translation origin is applied—added—to all (x, y) coordinates specified for rendering operations to locate the actual pixel coordinate on the drawing surface.

The origin translation does not apply to the clipping rectangle.

related function(s) `mpeos_gfxSetOrigin`

mpeos_gfxGetPaintMode

gets the current paint mode.

syntax `mpe_Error mpeos_gfxGetPaintMode(
 mpe_GfxContext context,
 mpe_GfxPaintMode * mode,
 uint32_t * data);`

| | | |
|---------------------|------------------------------|---|
| parameter(s) | <i>context</i> | specifies the handle to the context for which to get the current painting mode. <i>context</i> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <i>mode</i> | is an output pointer to the location where the current paint mode is stored. <code>mpe_GfxPaintMode</code> is described in the <i>mpe_GfxPaintMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | <code>MPE_GFX_XOR</code> | indicates an XOR bit operation. $Cd=Cd \oplus (Cs \oplus Cx)$; where Cd , Cs , and Cx are respectively destination, source, and XOR color values. Alpha value is ignored. |
| | <code>MPE_GFX_CLR</code> | indicates the following composition rule: $Cd=0$, $Ad=0$; where Ad is the destination alpha value. |
| | <code>MPE_GFX_SRC</code> | indicates the following composition rule: $Cd=Cs$, $Ad=As$; where As is the source alpha value. |
| | <code>MPE_GFX_SRCOVER</code> | indicates the following composition rule: $Cd=Cs+Cd*(1-As)$ and $Ad=As+Ad*(1-As)$. |
| | <code>MPE_GFX_DSTOVER</code> | indicates the following composition rule: $Cd=Cs*(1-Ad)+Cd$ and $Ad=As*(1-Ad)+Ad$. |
| | <code>MPE_GFX_SRCIN</code> | indicates the following composition rule: $Cd=Cs*Ad$ and $Ad=As*Ad$. |
| | <code>MPE_GFX_DSTIN</code> | indicates the following composition rule: $Cd=Cd*As$ and $Ad=Ad*As$. |
| | <code>MPE_GFX_SRCOUT</code> | indicates the following composition rule: $Cd=Cs*(1-Ad)$ and $Ad=As*(1-Ad)$. |

MPE_GFX_DSTOUT

indicates the following composition rule:
 $Cd=Cd*(1-As)$ and $Ad=Ad*(1-As)$.

MPE_GFX_DST

indicates the following composition rule:
 $Cd=Cd$ and $Ad=Ad$.

-
- ◆ **NOTE:** The alpha modulation constant is multiplied against the source alpha value to produce a new source alpha value for the composition. For example, if the modulation constant is 127 ($127/255 = 0.5$) and the source alpha constant is 127 ($127/255 = 0.5$), the effective source alpha value is 63.

data

*is an output pointer to the location where the current paint mode data parameter is stored. For exclusive or (XOR) *data* is the XOR color (an mpe_GfxColor value). For all other operations, *data* is the constant alpha modulation value, which is in the range from 0 to 255.*

value returned

If the call is successful, mpeos_gfxGetPaintMode() should return MPE_GFX_ERROR_NOERR. Otherwise, it should return the following error code:

MPE_GFX_ERROR_INVALID

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description

mpeos_gfxGetPaintMode() should get the current paint mode used for display rendering. The paint mode provides the manner in which the source and destination pixels are composited. The paint mode supports the 9 Porter-Duff alpha composition rules, plus an XOR rule corresponding to Abstract Windowing Toolkit semantics. The default rule is MPE_GFX_SRCOVER.

-
- ◆ **NOTE:** Rendering to the screen surface need only support XOR, SRC, SRCOVER, and CLR modes.

The semantics required for AWT by the XOR rule are as follows:

- ◆ When drawing operations are performed, pixels that are the current color are changed to the specified color, and vice versa.
- ◆ Pixels of colors other than those two colors are changed in an unpredictable but reversible manner; if the same figure is drawn twice, then all pixels are restored to their original values.

-
- ◆ **NOTE:** The preceding semantics are from the [JDK 1.1.8 API](#) documentation.

related function(s)

mpeos_gfxSetPaintMode

mpeos_gfxGetSurface

gets the surface to which the display is attached.

syntax mpe_Error mpeos_gfxGetSurface(
 mpe_GfxContext context,
 mpe_GfxSurface * surface);

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>context</i> | specifies the handle to the display context for which a surface handle is desired. <i>context</i> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <i>surface</i> | is an output pointer to the location where the surface handle for this context is stored. <code>mpe_GfxSurface</code> is described in the <i>mpe_GfxSurface</i> section. |

value returned If the call is successful, `mpeos_gfxGetSurface()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxGetSurface()` should get the surface to which the display context specified by *context* is attached. The display context was attached to the surface when the display context was created. Therefore, there is no `mpeos_gfxSetSurface()` function.

related function(s) `mpeos_gfxGetClipRect`
`mpeos_gfxGetColor`
`mpeos_gfxGetFont`
`mpeos_gfxGetOrigin`
`mpeos_gfxGetPaintMode`
`mpeos_gfxSetClipRect`
`mpeos_gfxSetColor`
`mpeos_gfxSetFont`
`mpeos_gfxSetOrigin`
`mpeos_gfxSetPaintMode`

mpeos_gfxSetClipRect

sets the display context's current clipping rectangle.

syntax `mpe_Error mpeos_gfxSetClipRect(
 mpe_GfxContext context,
 mpe_GfxRectangle * rect);`

parameter(s) `context` specifies the handle to the context for which to set the clipping rectangle. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`rect` is an input pointer to the location of the new rectangle used to define the clipping area for the context. `mpe_GfxRectangle` is described in the *mpe_GfxRectangle* section.

value returned If the call is successful, `mpeos_gfxSetClipRect()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID` indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxSetClipRect()` should set the current clipping rectangle that applies to all rendering operations. The rectangle is applied to the context's drawable surface, so the (x, y) coordinates are relative to (0, 0) the drawing surface.

The clipping rectangle for a context is defined as the intersection of the specified rectangle and the bounds of the drawable surface. It is legal for the clipping rectangle to extend beyond the defined surface—for example, you could have a negative x or negative y value, or a width or height that extends beyond the width and height of the surface. However, internally, this should be minimized to the intersection.

The origin translation does not apply to the set/get clipping rectangle.

related function(s) `mpeos_gfxGetClipRect`

mpeos_gfxSetColor

sets the current color used for display rendering operations.

syntax `mpe_Error mpeos_gfxSetColor(
 mpe_GfxContext context,
 mpe_GfxColor color);`

parameter(s) `context` specifies the handle to the context for which to set the color attributes. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`color` specifies the new color value to use for all subsequent rendering. `mpe_GfxColor` is described in the *mpe_GfxColor* section.

value returned If the call is successful, `mpeos_gfxSetColor()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxSetColor()` should set the current color attribute of the display context specified by `context`. The color is specified as an `mpe_GfxColor` value.

related function(s) `mpeos_gfxGetColor`

mpeos_gfxSetFont

sets the current font used for text rendering operations.

syntax `mpe_Error mpeos_gfxSetFont(
 mpe_GfxContext context,
 mpeH_GfxFont_t font);`

parameter(s) `context` specifies the handle to the context for which the font attribute should be set. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`font` specifies a handle to use for all subsequent text rendering. `font` is returned from a previous call to `mpeos_gfxFontCreate()` or `mpeos_gfxFontNew()`. `mpeH_GfxFont_t` is described in the *mpe_GfxFont* section.

value returned If the call is successful, `mpeos_gfxSetFont()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID` indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxSetFont()` should set the current font attribute of the display context specified by `context`.

related function(s) `mpeos_gfxGetFont`

mpeos_gfxSetOrigin

sets the display context's current translation origin.

syntax mpe_Error mpeos_gfxSetOrigin(
 mpe_GfxContext context,
 int32 x,
 int32 y);

| | | |
|---------------------|----------------|--|
| parameter(s) | context | specifies the handle to the context for which to set the translation origin. <i>context</i> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | x | specifies the x coordinate of the new translation origin, relative to the (0, 0) location of the associated surface. |
| | y | specifies the y coordinate of the new translation origin, relative to the (0, 0) location of the associated surface. |

value returned If the call is successful, `mpeos_gfxSetOrigin()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxSetOrigin()` should set the current translation origin used during rendering operations. The translation origin is applied—added—to all (x, y) coordinates specified for rendering operations to locate the actual pixel coordinate on the drawing surface.

The origin translation does not apply to the clipping rectangle.

related function(s) `mpeos_gfxGetOrigin`

mpeos_gfxSetPaintMode

sets the current paint mode.

syntax `mpe_Error mpeos_gfxSetPaintMode(
 mpe_GfxContext context,
 mpe_GfxPaintMode mode,
 uint32_t data);`

| | | |
|---------------------|------------------------------|--|
| parameter(s) | <i>context</i> | specifies the handle to the context for which the paint mode should be set. <i>context</i> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <i>mode</i> | specifies the new paint mode to be used for all subsequent operations. In all but the XOR cases, the source and destination colors (C_s and C_d , respectively) are considered to be premultiplied with their associated alphas (A_s and A_d , respectively). <code>mpe_GfxPaintMode</code> is described in the <i>mpe_GfxPaintMode</i> section. Currently, the following values are supported for <i>mode</i> : |
| | <code>MPE_GFX_XOR</code> | indicates an XOR bit operation. $C_d = C_d \oplus (C_s \oplus C_x)$; where C_d , C_s , and C_x are respectively destination, source, and XOR color values. Alpha value is ignored. |
| | <code>MPE_GFX_CLR</code> | indicates the following composition rule: $C_d = 0$, $A_d = 0$; where A_d is the destination alpha value. |
| | <code>MPE_GFX_SRC</code> | indicates the following composition rule: $C_d = C_s$, $A_d = A_s$; where A_s is the source alpha value. |
| | <code>MPE_GFX_SRCOVER</code> | indicates the following composition rule: $C_d = C_s + C_d * (1 - A_s)$ and $A_d = A_s + A_d * (1 - A_s)$. |
| | <code>MPE_GFX_DSTOVER</code> | indicates the following composition rule: $C_d = C_s * (1 - A_d) + C_d$ and $A_d = A_s * (1 - A_d) + A_d$. |
| | <code>MPE_GFX_SRCIN</code> | indicates the following composition rule: $C_d = C_s * A_d$ and $A_d = A_s * A_d$. |
| | <code>MPE_GFX_DSTIN</code> | indicates the following composition rule: $C_d = C_d * A_s$ and $A_d = A_d * A_s$. |
| | <code>MPE_GFX_SRCOUT</code> | indicates the following composition rule: $C_d = C_s * (1 - A_d)$ and $A_d = A_s * (1 - A_d)$. |

MPE_GFX_DSTOUT

indicates the following composition rule:
 $Cd=Cd*(1-As)$ and $Ad=Ad*(1-As)$.

MPE_GFX_DST

indicates the following composition rule:
 $Cd=Cd$ and $Ad=Ad$.

-
- ◆ **NOTE:** The alpha modulation constant is multiplied against the source alpha value to produce a new source alpha value for the composition. For example, if the modulation constant is 127 ($127/255 = 0.5$) and the source alpha constant is 127 ($127/255 = 0.5$), the effective source alpha value is 63.

data

specifies the paint mode data parameter to use for all subsequent rendering operations. For XOR, *data* is the XOR color (an `mpe_GfxColor` value). For all other operations, *data* is the constant alpha modulation value in the range of 0 to 255.

value returned

If the call is successful, `mpeos_gfxSetPaintMode()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

MPE_GFX_ERROR_INVALID

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description

`mpeos_gfxSetPaintMode()` should set the current paint mode used for display rendering. The paint mode provides the manner in which the source and destination pixels are composited. It supports the nine Porter-Duff alpha composition rules, plus an XOR rule corresponding to AWT semantics. The default rule is SRCOVER.

-
- ◆ **NOTE:** Rendering to the screen surface need only support XOR, SRC, SRCOVER, and CLR modes.

The semantics required for AWT by the XOR rule are as follows:

- ◆ When drawing operations are performed, pixels which are the current color are changed to the specified color, and vice versa.
- ◆ Pixels of colors other than those two colors are changed in an unpredictable but reversible manner; if the same figure is drawn twice, then all pixels are restored to their original values.

-
- ◆ **NOTE:** The preceding semantics are from the [JDK 1.1.8 API](#) documentation.

related function(s)

`mpeos_gfxGetPaintMode`

Drawing functions

All display operations render using an immediate mode. This means screen operations need to at least appear as if they are being performed directly on the frame buffer. All operations are subject to the current properties of the given display context. This includes using the current color, font, paint mode, origin, and clipping rectangle.

-
- ◆ **NOTE:** Outline (draw) operations follow the Java AWT semantics, whereby they draw an extra row/column of pixels (as the arguments to DrawRect/Ellipse/RoundRect specify a coordinate path, not the dimensions of the rectangle). So it is expected that FillRect(width, height) result in a filled rectangle width pixels wide and height pixels height; whereas DrawRect(width, height) results in an outlined rectangle width+1 pixels wide and height+1 pixels high.
-

The following drawing functions, which are defined in `mpeos_draw.h`, need to be supported:

- `mpeos_gfxBitBlt`
performs a bit block transfer of pixels from one drawing surface to another.
- `mpeos_gfxClearRect`
fills the rectangle with color.
- `mpeos_gfxDrawArc`
draws an outlined arc.
- `mpeos_gfxDrawEllipse`
draws an outlined ellipse.
- `mpeos_gfxDrawLine`
draws a line from one point to another.
- `mpeos_gfxDrawPolygon`
draws a polygon.
- `mpeos_gfxDrawPolyline`
draws a set of connected points.
- `mpeos_gfxDrawRect`
draws the outline of a rectangle.
- `mpeos_gfxDrawRoundRect`
draws an outlined rectangle with rounded corners.
- `mpeos_gfxDrawString`
draws the given string at the given baseline coordinates.
- `mpeos_gfxDrawString16`
draws the given UTF-16 string at the given baseline coordinates.
- `mpeos_gfxFillArc`
draws a filled arc.

`mpeos_gfxFillEllipse`
draws a filled ellipse.

`mpeos_gfxFillPolygon`
draws a filled polygon.

`mpeos_gfxFillRect`
draws a filled rectangle.

`mpeos_gfxFillRoundRect`
draws a filled rectangle with rounded corners.

`mpeos_gfxStretchBlt`
scales while performing a bit block transfer of data.

mpeos_gfxBitBlt

performs a bit block transfer of pixels from one drawing surface to another.

syntax

```
mpe_Error mpeos_gfxBitBlt(
    mpe_GfxContext dest,
    mpe_GfxSurface source,
    int32 dx,
    int32 dy,
    mpe_GfxRectangle * srect );
```

parameter(s) *dest* specifies the handle to the destination context for the pixel transfer. *mpe_GfxContext* is described in the *mpe_GfxContext* section.

source specifies the handle to the drawable surface that is the source for the pixel transfer. *mpe_GfxSurface* is described in the *mpe_GfxSurface* section.

dx, *dy* specify the destination coordinates within the destination context for the transfer.

srect is an input pointer to the source rectangle within the source surface for the transfer. *mpe_GfxRectangle* is described in the *mpe_GfxRectangle* section.

value returned If the call is successful, *mpeos_gfxBitBlt()* should return *MPE_GFX_ERROR_NOERR*. Otherwise, it should return the following error code:

MPE_GFX_ERROR_OSERR
indicates an operating system error occurred.

description *mpeos_gfxBitBlt()* should perform a bit block transfer of pixels from one drawing surface to another. *mpeos_gfxBitBlt()* should perform a one-to-one copy of pixels from the source to the destination. This transfer is subject to the current paint mode (for example, alpha composition rule).

related function(s) *mpeos_gfxStretchBlt*

mpeos_gfxClearRect

fills the rectangle with color.

syntax `mpe_Error mpeos_gfxClearRect
 mpe_GfxContext ctx,
 mpe_GfxRectangle * rect,
 mpe_GfxColor color);`

| | | |
|---------------------|--------------------|--|
| parameter(s) | <code>ctx</code> | specifies the context with which to render. <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>rect</code> | is an input pointer to the path taken in rendering the interior of the rectangle. <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |
| | <code>color</code> | specifies the color used to fill in the rectangle. <code>mpe_GfxColor</code> is described in the <i>mpe_GfxColor</i> section. |

value returned If the call is successful, `mpeos_gfxClearRect()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxClearRect` should fill the specified rectangle with the given color. The left and right edges of the rectangle are at `x` and `x + width - 1`. The top and bottom edges are at `y` and `y + height - 1`. The resulting rectangle covers an area `width` pixels wide by `height` pixels tall. The rectangle is filled using the given color rather than the graphics context's current color.

related function(s) `mpeos_gfxFillRect`

mpeos_gfxDrawArc

draws an outlined arc.

syntax `mpe_Error mpeos_gfxDrawArc(mpe_GfxContext context, mpe_GfxRectangle * bounds, int32 startAngle, int32 endAngle);`

parameter(s) `context` specifies the handle to the context on which to draw the outlined arc. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`bounds` is an input pointer to the boundary to use when drawing the arc. `mpe_GfxRectangle` is described in the *mpe_GfxRectangle* section.

`startAngle` specifies the beginning angle.

`endAngle` specifies the angular extent of the arc, relative to `startAngle`.

value returned If the call is successful, `mpeos_gfxDrawArc()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxDrawArc()` draws the outline corresponding to the elliptical arc. The draw operation draws the single-pixel outer edge of the arc.

Angles are interpreted such that 0° corresponds to the 3 o'clock position. A positive value indicates a counter-clockwise rotation; negative indicates clockwise.

The resulting arc covers an area `bounds.width+1` pixels by `bounds.height+1` pixels. The center of the arc is the center of the bounding rectangle. A line at a 45° angle always points to a corner of the rectangle (for example, if the rectangle is longer in one direction, the angles are skewed along the longer axis).

related function(s) `mpeos_gfxFillArc`

mpeos_gfxDrawEllipse

draws an outlined ellipse.

syntax `mpe_Error mpeos_gfxDrawEllipse(
 mpe_GfxContext context,
 mpe_GfxRectangle * bounds);`

parameter(s) `context` specifies the handle to the context on which to draw the ellipse. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`bounds` is an input pointer to the boundary of the ellipse. `mpe_GfxRectangle` is described in the *mpe_GfxRectangle* section.

value returned If the call is successful, `mpeos_gfxDrawEllipse()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxDrawEllipse()` should draw an outlined ellipse. The horizontal and vertical diameters are `bounds.width+1` and `bounds.height+1` pixels, respectively.

related function(s) `mpeos_gfxFillEllipse`

mpeos_gfxDrawLine

draws a line from one point to another.

syntax `mpe_Error mpeos_gfxDrawLine(
 mpe_GfxContext context,
 int32 x1,
 int32 y1,
 int32 x2,
 int32 y2);`

parameter(s) `context` specifies the handle to the context to use for drawing the line. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`x1` specifies the x coordinate of the first endpoint that defines the line.

`y1` specifies the y coordinate of the first endpoint that defines the line.

`x2` specifies the x coordinate of the second endpoint that defines the line.

`y2` specifies the y coordinate of the second endpoint that defines the line.

value returned If the call is successful, `mpeos_gfxDrawLine()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxDrawLine()` should draw a single-pixel line from the endpoint specified by (x_1, y_1) to the endpoint specified by (x_2, y_2) . The line is drawn using the current color.

This operation is subject to the current paint mode.

related function(s) `mpeos_gfxDrawPolyline`

mpeos_gfxDrawPolygon

draws a polygon.

syntax `mpe_Error mpeos_gfxDrawPolygon(mpe_GfxContext context, int32 * xCoords, int32 * yCoords, int32 nCoords);`

| | | |
|---------------------|----------------------|---|
| parameter(s) | <code>context</code> | specifies the handle to the context on which to draw the polygon. <code>context</code> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>xCoords</code> | is an input pointer to the location of an array of size <code>nCoords</code> specifying the <code>xCoords</code> coordinates. |
| | <code>yCoords</code> | is an input pointer to the location of an array of size <code>nCoords</code> specifying the <code>yCoords</code> coordinates. |
| | <code>nCoords</code> | specifies the size of the arrays specified by <code>xCoords</code> and <code>yCoords</code> . |

value returned If the call is successful, `mpeos_gfxDrawPolygon()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxDrawPolygon()` should draw an outlined polygon using the path defined with the (`xCoords`, `yCoords`) coordinates. The first and last endpoints ((`xCoords[0]`,`yCoords[0]`) and (`xCoords[nCoords-1]`,`yCoords[nCoords-1]`), respectively) are connected if not already specified.

related function(s) `mpeos_gfxDrawPolyline`
`mpeos_gfxFillPolygon`

mpeos_gfxDrawPolyline

draws a set of connected points.

syntax `mpe_Error mpeos_gfxDrawPolyline(mpe_GfxContext context, int32 * xCoords, int32 * yCoords, int32 nCoords);`

| | | |
|---------------------|----------------------|--|
| parameter(s) | <code>context</code> | specifies the handle to the context on which to draw the polyline. <code>context</code> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>xCoords</code> | is an input pointer to the location of an array of size <code>nCoords</code> specifying the <code>xCoords</code> coordinates. |
| | <code>yCoords</code> | is an input pointer to the location of an array of size <code>nCoords</code> specifying the <code>yCoords</code> coordinates. |
| | <code>nCoords</code> | specifies the size of the arrays specified by <code>xCoords</code> and <code>yCoords</code> . |

value returned If the call is successful, `mpeos_gfxDrawPolyline()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxDrawPolyline()` should draw several single-pixels as defined by the `xCoords` and `yCoords` arrays using the current color. This operation is equivalent to performing several individual `DrawLine` calls such as the following:

```
for (int i = 0; i < nCoords-1; ++i) {
    mpe_gfxDrawLine(content,
                     xCoords[i], yCoords[i],
                     xCoords[i+1], yCoords[i+1]);
}
```

related function(s) `mpeos_gfxDrawLine`
`mpeos_gfxDrawPolygon`

mpeos_gfxDrawRect

draws the outline of a rectangle.

syntax `mpe_Error mpeos_gfxDrawRect(
 mpe_GfxContext context,
 mpe_GfxRectangle * rect);`

| | | |
|---------------------|----------------------|---|
| parameter(s) | <code>context</code> | specifies the handle to the context on which to draw the rectangle. <code>context</code> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>rect</code> | is an input pointer to the boundary of the rectangle to draw. <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |

value returned If the call is successful, `mpeos_gfxDrawRect()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxDrawRect()` should draw the outline of the rectangle specified by `rect` on the context specified by `context`. The left and right edges of the rectangle are at `x` and `x+width`. The top and bottom edges are at `y` and `y+height`. The rectangle is drawn using the graphics context's current color.

-
- ◆ **NOTE:** The drawn rectangle bounds are +1 in both the horizontal and the vertical directions.
-

This operation is subject to the current paint mode.

related function(s) `mpeos_gfxFillRect`

mpeos_gfxDrawRoundRect

draws an outlined rectangle with rounded corners.

syntax `mpe_Error mpeos_gfxDrawRoundRect(mpe_GfxContext context, mpe_GfxRectangle * rect, int32 arcWidth, int32 arcHeight);`

| | | |
|---------------------|------------------------|---|
| parameter(s) | <code>context</code> | specifies the handle to the context on which to draw the outlined rectangle with rounded corners. <code>context</code> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>rect</code> | is an input pointer to the boundary to use when drawing the rectangle with rounded corners. <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |
| | <code>arcWidth</code> | specifies the horizontal diameter of the arc at the four corners. |
| | <code>arcHeight</code> | specifies the vertical diameter of the arc at the four corners. |

value returned If the call is successful, `mpeos_gfxDrawRoundRect()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:
`MPE_GFX_ERROR_OSERR`
 indicates an operating system error occurred.

description `mpeos_gfxDrawRoundRect()` draws an outlined rectangle with rounded corners. The horizontal and vertical diameters are `bounds.width+1` and `bounds.height+1`, respectively.

`arcWidth` and `arcHeight` specify the horizontal and vertical diameters (respectively) of the arcs drawn at each of the four corners.

Figure GFX-3 shows how the parameters are interpreted for the draw function.

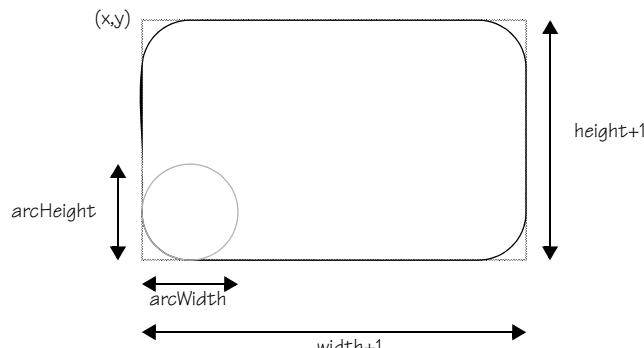


FIGURE GFX-3:

Interpretation of the parameters for the draw function

related function(s) mpeos_gfxFillRoundRect

mpeos_gfxDrawString

draws the given string at the given baseline coordinates.

syntax `mpe_Error mpeos_gfxDrawString(
 mpe_GfxContext context,
 int32 x,
 int32 y,
 const char * buffer,
 int32 length);`

parameter(s) `context` specifies the handle to the context on which to draw the string. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`x` specifies the x coordinate for the starting baseline.

`y` specifies the y coordinate for the starting baseline.

`buffer` is an input pointer to the buffer to render.

`length` specifies the number of characters to render.

value returned If the call is successful, `mpeos_gfxDrawString()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxDrawString()` should draw the given byte at the baseline coordinates specified by `x`, `y` using the context specified by `context`. A byte string is interpreted as an array of bytes corresponding to the first 256 encodings of Unicode Transformation Format (UTF)-16 (that is, International Organization for Standardization (ISO) 8859-1).

The current font, color, clipping, and paint mode apply. The given coordinate is interpreted as the left-most extent of the baseline as shown in Figure GFX-4.

FIGURE GFX-4:
Baseline coordinates



`length` indicates the number of bytes to use from the given array to render, up to but not including any terminating NULL character. This directly maps to the number of glyphs to display.

-
- ◆ **NOTE:** A UTF-8 version of the DrawString routine is not specified. Strings are encoded in Java using UTF-16, so supporting it directly is necessary to avoid information loss. The AWT graphics object also supports a drawBytes() method that takes an array of bytes representing the first 256 encodings of UTF-16. The AWT implementation includes support to emulate full Unicode font support using multiple platform fonts; this is done using drawBytes(). The only way to get UTF-8 characters is via Java Native Interface (JNI) GetStringUTFChars() function, as such, support for UTF-8 is not necessary in the MPE graphics API.
-

related function(s) mpeos_gfxDrawString16

mpeos_gfxDrawString16

draws the given UTF-16 string at the given baseline coordinates.

syntax `mpe_Error mpeos_gfxDrawString16(
 mpe_GfxContext context,
 int32 x,
 int32 y,
 const mpe_GfxWchar * buffer,
 int32 length);`

parameter(s) `context` specifies the handle to the context on which to draw the UTF-16 string. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`x` specifies the x coordinate for the starting baseline.

`y` specifies the y coordinate for the starting baseline.

`buffer` is an input pointer to the UTF-16 buffer to render. `mpe_GfxWchar` is described in the *mpe_GfxWchar* section.

`length` specifies the number of characters to render.

value returned If the call is successful, `mpeos_gfxDrawString16()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxDrawString16()` should draw the given UTF-16 string at the baseline coordinates specified by `x`, `y` using the context specified by `context`. A byte string is interpreted as an array of bytes corresponding to the first 256 encodings of UTF-16 (that is, ISO 8859-1). The current font, color, clipping, and paint mode apply. The given coordinate is interpreted as the left-most extent of the baseline as shown in Figure GFX-5.

FIGURE GFX-5:
Baseline coordinates



The diagram shows the text "Hello, world!" centered on a horizontal line labeled "(x,y)". The text is rendered in a black sans-serif font.

`length` indicates the number of bytes to use from the given array to render, up to but not including any terminating NULL character. This directly maps to the number of glyphs to display.

-
- ◆ **NOTE:** A UTF-8 version of the DrawString routine is not specified. Strings are encoded in Java using UTF-16, so supporting it directly is necessary to avoid information loss. The AWT graphics object also supports a drawBytes() method that takes an array of bytes representing the first 256 encodings of UTF-16. The AWT implementation includes support to emulate full Unicode font support using multiple platform fonts; this is done using drawBytes(). The only way to get UTF-8 characters is via JNI GetStringUTFChars() function, as such, support for UTF-8 is not necessary in the MPE graphics API.
-

related function(s) [mpeos_gfxDrawString](#)

mpeos_gfxFillArc

draws a filled arc.

syntax `mpe_Error mpeos_gfxFillArc(mpe_GfxContext context, mpe_GfxRectangle * bounds, int startAngle, int endAngle);`

| | | |
|---------------------|-------------------|--|
| parameter(s) | <i>context</i> | specifies the handle to the context on which to draw the filled arc. <i>context</i> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <i>bounds</i> | is an input pointer to the boundary to use when drawing the filled arc with rounded corners. <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |
| | <i>startAngle</i> | specifies the beginning angle. |
| | <i>endAngle</i> | specifies the angular extent of the arc, relative to <i>startAngle</i> . |

value returned If the call is successful, `mpeos_gfxFillArc()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxFillArc()` should draw a pie corresponding to an elliptical arc. The fill operation fills the internal area of the arc.

Angles are interpreted such that 0° corresponds to the 3 o'clock position. A positive value indicates a counter-clockwise rotation; negative indicates clockwise.

The resulting arc covers an area `bounds.width+1` pixels by `bounds.height+1` pixels. The center of the arc is the center of the bounding rectangle. A line at a 45° angle always points to a corner of the rectangle (for example, if the rectangle is longer in one direction, the angles are skewed along the longer axis).

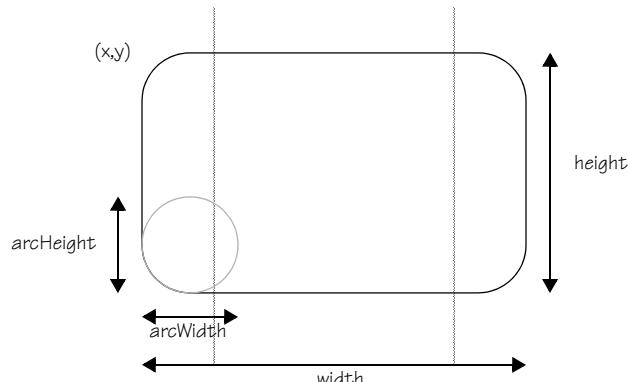


FIGURE GFX-6:

Interpretation of the parameters for the fill function

related function(s) mpeos_gfxDrawArc

mpeos_gfxFillEllipse

draws a filled ellipse.

syntax mpe_Error mpeos_gfxFillEllipse(
 mpe_GfxContext *context*,
 mpe_GfxRectangle * *bounds*);

parameter(s) *context* specifies the handle to the context on which to draw the filled ellipse. *context* is returned by a previous call to mpeos_gfxContextNew() or mpeos_gfxContextCreate(). mpe_GfxContext is described in the *mpe_GfxContext* section.

bounds is an input pointer to the boundary of the ellipse. mpe_GfxRectangle is described in the *mpe_GfxRectangle* section.

value returned If the call is successful, mpeos_gfxFillEllipse() should return MPE_GFX_ERROR_NOERR. Otherwise, it should return the following error code:

MPE_GFX_ERROR_OSERR
indicates an operating system error occurred.

description mpeos_gfxFillEllipse() should draw a filled ellipse. The horizontal and vertical diameters are *bounds.width* and *bounds.height*, respectively.

related function(s) mpeos_gfxDrawEllipse

mpeos_gfxFillPolygon

draws a filled polygon.

syntax `mpe_Error mpeos_gfxFillPolygon(mpe_GfxContext context, int32 * xCoords, int32 * yCoords, int32 nCoords);`

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>context</i> | specifies the handle to the context on which to draw the filled polygon. <i>context</i> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <i>xCoords</i> | is an input pointer to the location of an array of size <i>nCoords</i> specifying the x coordinates. |
| | <i>yCoords</i> | is an input pointer to the location of an array of size <i>nCoords</i> specifying the y coordinates. |
| | <i>nCoords</i> | specifies the size of the arrays specified by <i>xCoords</i> and <i>yCoords</i> . |

value returned If the call is successful, `mpeos_gfxFillPolygon()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxFillPolygon()` should draw a filled polygon using the path defined with the (x, y) coordinates as specified by *xCoords* and *yCoords*. The first and last endpoints ($(xCoords[0], yCoords[0])$ and $(xCoords[nCoords-1], yCoords[nCoords-1])$, respectively) are connected if not already specified.

The area filled by the fill operation is defined by an even-odd (also known as the alternating) rule such that areas where the polygon intersects itself may be filled or unfilled.

related function(s) `mpeos_gfxDrawPolygon`

mpeos_gfxFillRect

draws a filled rectangle.

syntax `mpe_Error mpeos_gfxFillRect(
 mpe_GfxContext context,
 mpe_GfxRectangle * rect);`

parameter(s) `context` specifies the handle to the context on which to fill the rectangle. `context` is returned by a previous call to `mpeos_gfxContextNew()` or `mpeos_gfxContextCreate()`. `mpe_GfxContext` is described in the *mpe_GfxContext* section.

`rect` is an input pointer to the boundary of the rectangle to fill. `mpe_GfxRectangle` is described in the *mpe_GfxRectangle* section.

value returned If the call is successful, `mpeos_gfxFillRect()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxFillRect()` should fill the rectangle specified by `rect`. The left and right edges of the rectangle are at `x` and `x+width-1`. The top and bottom edges are at `y` and `y+height-1`. The resulting rectangle covers an area `width` pixels wide by `height` pixels tall. The rectangle is filled using the graphics context's current color.

This operation is subject to the current paint mode.

related function(s) `mpeos_gfxDrawRect`

mpeos_gfxFillRoundRect

draws a filled rectangle with rounded corners.

syntax `mpe_Error mpeos_gfxFillRoundRect(mpe_GfxContext context, mpe_GfxRectangle * rect, int32 arcWidth, int32 arcHeight);`

| | | |
|---------------------|------------------------|---|
| parameter(s) | <code>context</code> | specifies the handle to the context on which to draw the filled rectangle with rounded corners. <code>context</code> is returned by a previous call to <code>mpeos_gfxContextNew()</code> or <code>mpeos_gfxContextCreate()</code> . <code>mpe_GfxContext</code> is described in the <i>mpe_GfxContext</i> section. |
| | <code>rect</code> | is an input pointer to the boundary to use when drawing the filled rectangle with rounded corners. <code>mpe_GfxRectangle</code> is described in the <i>mpe_GfxRectangle</i> section. |
| | <code>arcWidth</code> | specifies the horizontal diameter of the arc at the four corners. |
| | <code>arcHeight</code> | specifies the vertical diameter of the arc at the four corners. |

value returned If the call is successful, `mpeos_gfxFillRect()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:
`MPE_GFX_ERROR_OSERR`
 indicates an operating system error occurred.

description `mpeos_gfxFillRoundRect()` should draw a filled rectangle with rounded corners. The horizontal and vertical diameters are `bounds.width` and `bounds.height`, respectively.

`arcWidth` and `arcHeight` specify the horizontal and vertical diameters (respectively) of the arcs drawn at each of the four corners.

Figure GFX-7 shows how the parameters are interpreted for the draw function.

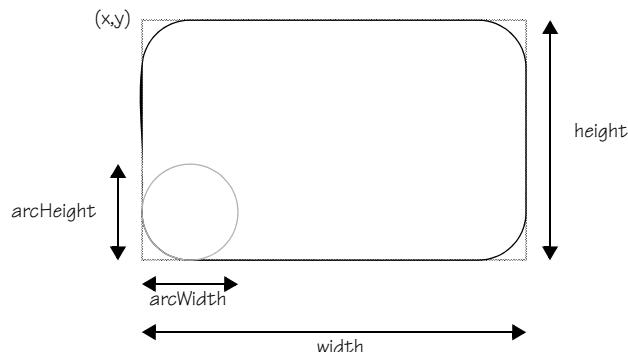


FIGURE GFX-7:

Interpretation of the parameters for the fill function

related function(s) `mpeos_gfxDrawRoundRect`

mpeos_gfxStretchBlt

scales while performing a bit block transfer of data.

syntax

```
mpe_Error mpeos_gfxStretchBlt(
    mpe_GfxContext dest,
    mpe_GfxSurface source,
    mpe_GfxRectangle * drect,
    mpe_GfxRectangle * srect );
```

| | | |
|---------------------|---------------|--|
| parameter(s) | <i>dest</i> | specifies the handle to the destination context to use for the pixel transfer. <i>mpe_GfxContext</i> is described in the <i>mpe_GfxContext</i> section. |
| | <i>source</i> | specifies the drawable surface that is the source for the pixel transfer. <i>mpe_GfxSurface</i> is described in the <i>mpe_GfxSurface</i> section. |
| | <i>drect</i> | is an input pointer to the destination rectangle within the destination context for the transfer. <i>mpe_GfxRectangle</i> is described in the <i>mpe_GfxRectangle</i> section. |
| | <i>srect</i> | is an input pointer to the source rectangle within the source surface for the transfer. <i>mpe_GfxRectangle</i> is described in the <i>mpe_GfxRectangle</i> section. |

value returned If the call is successful, `mpeos_gfxStretchBlt()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxStretchBlt()` should perform a bit block transfer of pixels from one drawing surface to another. It should also perform a scaling operation as part of the data transfer.

This transfer is subject to the current paint mode (for example, the alpha composition rule).

`mpeos_gfxStretchBlt()` should perform the scaling operation such that the image is scaled in the horizontal and vertical directions using the following scaling ratios:

$$\begin{aligned} \text{scaleH} &= \text{drect.width}/\text{srect.width} \\ \text{scaleV} &= \text{drect.height}/\text{srect.height} \end{aligned}$$

Any appropriate conversions are implicitly applied (for example, from the source to destination surface formats) by the bit block transfer operations. However, such conversions should be minimized by the following:

- ◆ minimizing the number of formats for a given implementation
- ◆ creating compatible surfaces

related function(s) `mpeos_gfxBitBlt`

Font functions

The font functions implement the interface for creating a font and providing various matrix information on a given font. A font is created from a factory, which has a font descriptor that matches the desired font name, style, and size. If an exact match is not found, the font factory tries to find a match as close as possible to the desired font. The following font functions, which are defined in `mpeos_font.h`, need to be supported:

- `mpeos_gfxFontDelete`
deletes the specified font.
- `mpeos_gfxFontGetList`
gets the head of the font list.
- `mpeos_gfxFontHasCode`
determines whether a specific code is supported by the given font.
- `mpeos_gfxFontNew`
creates a new font.
- `mpeos_gfxGetFontMetrics`
gets detailed information about a font.
- `mpeos_gfxGetCharWidth`
gets the width of the specified character in the specified font.
- `mpeos_gfxGetStringWidth`
gets the width of the specified byte.
- `mpeos_gfxGetString16Width`
gets the width of the specified UTF-16 string.

mpeos_gfxFontDelete

deletes the specified font.

syntax `mpe_Error mpeos_gfxFontDelete(mpe_GfxFont font);`

parameter(s) `font` specifies a handle to the font reference to delete. `font` is returned from a previous call to `mpeos_gfxFontCreate()` or `mpeos_gfxFontNew()`. `mpe_GfxFont` is described in the *mpe_GfxFont* section.

value returned If the call is successful, `mpeos_gfxFontDelete()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxFontDelete()` should delete the font specified by `font`. A font is physically deleted only when its reference count is zero.

◆ **NOTE:** The default system font cannot be deleted.

related function(s) `mpeos_gfxFontNew`

mpeos_gfxFontGetList

gets the head of the font list.

syntax mpe_Error mpeos_gfxFontGetList (mpe_GfxFontDesc ** *description*);

parameter(s) *description* is an output pointer to the address of the head of the list. *desc->next* is the first element in the list. *mpe_GfxFontDesc* is described in the *mpe_GfxFontDesc* section.

value returned If the call is successful, *mpeos_gfxFontGetList()* should return *MPE_GFX_ERROR_NOERR*.

description *mpeos_gfxFontGetList()* should get the head of the font list of the system factory font.

related function(s) none

mpeos_gfxFontHasCode

determines whether a specific code is supported by the given font.

syntax mpe_Error mpeos_gfxFontHasCode(
 mpe_GfxFont *font*,
 mpe_GfxWchar *code*);

parameter(s) *font* specifies a handle to a graphics font. *font* is returned from a previous call to `mpeos_gfxFontCreate()` or `mpeos_gfxFontNew()`. `mpe_GfxFont` is described in the *mpe_GfxFont* section.

code indicates the code is in the font. `mpe_GfxWchar` is described in the *mpe_GfxWchar* section.

value returned If the call is successful, `mpeos_gfxFontHasCode()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
 indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_FALSE`
 indicates the font does not support the *code*.

description `mpeos_gfxFontHasCode()` determines whether the font specified by *font* supports a particular code (that is, whether it has a glyph for the given font).

related function(s) none

mpeos_gfxFontNew

creates a new font.

syntax mpe_Error mpeos_gfxFontNew(
 mpe_GfxFontFactory *ff*,
 const mpe_GfxWchar * *name*,
 const uint32_t *nameLength*,
 mpe_GfxFontStyle *style*,
 int32 *size*,
 mpe_GfxFont * *font*);

parameter(s) *ff*

specifies the font factory to use for creating the font. If *ff* is NULL, the local system fonts should be used.
mpe_GfxFontFactory is described in the *mpe_GfxFontFactory* section.

name

is an input pointer to the UTF-16 name of the desired platform font. *mpe_GfxWchar* is described in the *mpe_GfxWchar* section.

nameLength

indicates the number of characters in the name.

style

specifies the font style. *mpe_GfxFontStyle* is described in the *mpe_GfxFontStyle* section. Currently, the following values are supported for *style*:

MPE_GFX_PLAIN

specifies the font does not have bold or italics formatting.

MPE_GFX_BOLD

specifies the font is bold.

MPE_GFX_ITALIC

specifies the font is italic.

MPE_GFX_BOLD_ITALIC

specifies the font is bold and italic.

size

specifies the desired pixel size for the font.

font

is an output pointer to the location where the font reference handle is to be stored when the call successfully completes. *mpe_GfxFont* is described in the *mpe_GfxFont* section.

value returned

If the call is successful, *mpeos_gfxFontNew()* should return MPE_GFX_ERROR_NOERR. Otherwise, it should return one of the following error codes:

MPE_GFX_ERROR_INVALID

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

GFX_ERR_NOFONT

specifies the font cannot be created. This may be returned for a non-system font factory.

description `mpeos_gfxFontNew()` should create a new reference to the specified font using the optional font factory. If the font factory specified in `ff` is not `NULL`, this factory should be used to construct the font. If `ff` is `NULL`, then the local system font is being referenced.

When the implicit system font factory is used (that is, `ff` is `NULL`), no error should ever be returned. If the desired font cannot be found or created, then a suitable replacement should be used. A suitable replacement is implementation dependent, but should at least be the default system font (that is, Tiresias PLAIN 26) and preferably as close to the requested size and style as possible.

related function(s) `mpeos_gfxFontDelete`

mpeos_gfxGetFontMetrics

gets detailed information about a font.

syntax mpe_Error mpeos_gfxGetFontMetrics(
 mpe_GfxFont *font*,
 mpe_GfxFontMetrics * *metrics*);

| | | |
|---------------------|-------------------------|---|
| parameter(s) | <i>font</i> | specifies the handle to the font for which to acquire metrics. <i>font</i> is returned from a previous call to <code>mpeos_gfxFontCreate()</code> or <code>mpeos_gfxFontNew()</code> . <code>mpe_GfxFont</code> is described in the <i>mpe_GfxFont</i> section. |
| | <i>metrics</i> | is an output pointer to the location where the font metrics information is to be stored when the call completes successfully. <code>mpe_GfxFontMetrics</code> is described in the <i>mpe_GfxFontMetrics</i> section. Currently, the following values are supported for <i>metrics</i> : |
| | <code>height</code> | indicates the total height in pixels of the font. This is the sum of <code>ascent</code> + <code>descent</code> + <code>leading</code> . |
| | <code>ascent</code> | indicates the amount of the font above the baseline. |
| | <code>descent</code> | indicates the amount of the font below the baseline. |
| | <code>leading</code> | indicates the interline spacing expected to be reserved between the <code>descent</code> and <code>ascent</code> of adjacent lines. |
| | <code>maxadvance</code> | indicates the maximum advance width of any character in this font. <code>maxadvance</code> may be -1 if the maximum advance width is not known. |
| | <code>maxascent</code> | indicates the maximum ascent for any code in this font. |
| | <code>maxdescent</code> | indicates the maximum descent for any code in this font. |
| | <code>first_char</code> | indicates the first character of the font character map. |
| | <code>last_char</code> | indicates the last character of the font character map. |

value returned If the call is successful, `mpeos_gfxGetFontMetrics()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxGetFontMetrics()` should retrieve low-level metrics information about the given font. All measurements are in pixels. Unknown metrics are set to `MPE_GFX_UNKNOWN`.

- ◆ **NOTE:** All approximations should be rounded up.

- ◆ **NOTE:** The definitions of these metrics should match that required by Java.

Figure GFX-8 provides an example of these metrics.

FIGURE GFX-8:
Font metrics example.



related function(s)

`mpeos_gfxGetCharWidth`
`mpeos_gfxGetStringWidth`
`mpeos_gfxGetString16Width`

mpeos_gfxGetCharWidth

gets the width of the specified character in the specified font.

syntax `mpe_Error mpeos_gfxGetCharWidth(
 mpe_GfxFont font,
 mpe_GfxWchar char,
 int32 * width);`

| | | |
|---------------------|--------------|--|
| parameter(s) | <i>font</i> | specifies a handle to a font used for the width calculation. <i>font</i> is returned from a previous call to <code>mpeos_gfxFontCreate()</code> or <code>mpeos_gfxFontNew()</code> . <code>mpe_GfxFont</code> is described in the <i>mpe_GfxFont</i> section. |
| | <i>char</i> | specifies the character with which to calculate the width. <code>mpe_GfxWchar</code> is described in the <i>mpe_GfxWchar</i> section. |
| | <i>width</i> | is an output pointer to the location where the calculated width is to be stored. |

value returned If the call is successful, `mpeos_gfxGetCharWidth()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxGetCharWidth()` should calculate and return the width of the character specified by *char* in the font specified by *font*. This calculation does not take kerning into account.

◆ **NOTE:** All approximations should be rounded up.

related function(s) `mpeos_gfxGetFontMetrics`
`mpeos_gfxGetStringWidth`
`mpeos_gfxGetString16Width`

mpeos_gfxGetStringWidth

gets the width of the specified byte.

syntax `mpe_Error mpeos_gfxGetStringWidth(mpe_GfxFont font, const char * string, int32 length, int32 * width);`

| | | |
|---------------------|---------------|--|
| parameter(s) | <i>font</i> | specifies a handle to the font used for the width calculation. <i>font</i> is returned from a previous call to <code>mpeos_gfxFontCreate()</code> or <code>mpeos_gfxFontNew()</code> . <code>mpe_GfxFont</code> is described in the <i>mpe_GfxFont</i> section. |
| | <i>string</i> | is an input pointer to the string buffer for which to calculate the width. |
| | <i>length</i> | specifies the number of characters to calculate width for, up to but not including a terminating NULL character. |
| | <i>width</i> | is an output pointer to the location where calculated width is to be stored. |

value returned If the call is successful, `mpeos_gfxGetStringWidth()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxGetStringWidth()` should calculate and return the width of the given string in pixels in the specified font. The calculation takes kerning into account.

◆ **NOTE:** All approximations should be rounded up.

related function(s) `mpeos_gfxGetFontMetrics`
`mpeos_gfxGetCharWidth`
`mpeos_gfxGetString16Width`

mpeos_gfxGetString16Width

gets the width of the specified UTF-16 string.

syntax `mpe_Error mpeos_gfxGetString16Width(
 mpe_GfxFont font,
 const mpe_GfxWchar * string,
 int32 length,
 int32 * width);`

| | | |
|---------------------|---------------|--|
| parameter(s) | <i>font</i> | specifies a handle to the font used for the width calculation. <i>font</i> is returned from a previous call to <code>mpeos_gfxFontCreate()</code> or <code>mpeos_gfxFontNew()</code> . <code>mpe_GfxFont</code> is described in the <i>mpe_GfxFont</i> section. |
| | <i>string</i> | is an input pointer to the UTF-16 string buffer for which to calculate a width. <code>mpe_GfxWchar</code> is described in the <i>mpe_GfxWchar</i> section. |
| | <i>length</i> | specifies the number of characters to calculate width for, up to but not including a terminating NULL character. |
| | <i>width</i> | is an output pointer to the location where calculated width is to be stored. |

value returned If the call is successful, `mpeos_gfxGetString16Width()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxGetString16Width()` should calculate and return the width of the given UTF-16 string in pixels in the specified font. The calculation takes kerning into account.

◆ **NOTE:** All approximations should be rounded up.

related function(s) `mpeos_gfxGetFontMetrics`
`mpeos_gfxGetCharWidth`
`mpeos_gfxGetStringWidth`

Font-factory functions

The font-factory implementation creates and deletes font factories, and links fonts to an existing font factory. The font factory uses its linked list of associated fonts to find a match as close as possible to the desired font name, style, and size.

-
- ◆ **NOTE:** Each font factory created is associated with one source of fonts. For example, if the fonts are being loaded from a specific URL, then this source of fonts would have its own font factory. This is encapsulated within the Java code.
-

The following font factory functions, which are defined in the `mpeos_fontfact.h`, need to be supported:

`mpeos_gfxFontFactoryAdd`
adds a font to the font factory.

`mpeos_gfxFontFactoryDelete`
deletes a font factory.

`mpeos_gfxFontFactoryNew`
creates and initializes a new font factory.

mpeos_gfxFontFactoryAdd

adds a font to the font factory.

syntax mpe_Error mpeos_gfxFontFactoryAdd(
 mpe_GfxFontFactory *fontFactory*,
 mpe_GfxFontDesc * *description*);

parameter(s) *fontFactory* specifies the font factory in which to add the font.

fontFactory is returned by a previous call to
mpeos_gfxFontFactoryNew(). *mpe_GfxFontFactory* is
described in the *mpe_GfxFontFactory* section.

description is an input pointer to the font to add to the font factory. Two
formats for the description are accepted, one with data and
data-size only (keys off of NULL name) and one with all the
information included. *mpe_GfxFontDesc* is defined in the
mpe_GfxFontDesc section.

value returned If the call is successful, mpeos_gfxFontFactoryAdd() should return
MPE_GFX_ERROR_NOERR. Otherwise, it should return one of the following
error codes:

MPE_GFX_ERROR_INVALID

indicates an invalid parameter (for example, out-of-range
values, null pointers, unknown values, etc.).

MPE_GFX_ERROR_UNKNOWN

indicates a generic error.

description mpeos_gfxFontFactoryAdd() should add a new font to the font factory.

related functions mpeos_gfxFontFactoryDelete
mpeos_gfxFontFactoryNew

mpeos_gfxFontFactoryDelete

Deletes a font factory.

syntax mpe_Error mpeos_gfxFontFactoryDelete(
 mpe_GfxFontFactory * *fontFactory*);

parameter(s) *fontFactory* is an input pointer to the location of the font factory to delete. *fontFactory* is returned by a previous call to `mpeos_gfxFontFactoryNew()`. `mpe_GfxFontFactory` is described in the *mpe_GfxFontFactory* section.

value returned If the call is successful, `mpeos_gfxFontFactoryDelete()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return the following error code:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description `mpeos_gfxFontFactoryDelete()` should delete an existing font factory if the global font count is zero (for example, there is no more font produced from the factory). If the count is non-zero, the factory is flagged as to-be-deleted and should be deleted after the last produced font has been deleted.

related functions `mpeos_gfxFontDelete`
`mpeos_gfxFontFactoryAdd`
`mpeos_gfxFontFactoryNew`

mpeos_gfxFontFactoryNew

creates and initializes a new font factory.

syntax `mpe_Error mpeos_gfxFontFactoryNew(
 mpe_GfxFontFactory * fontFactory);`

parameter(s) `fontFactory` is an input pointer to the location of the new font factory.
`mpe_GfxFontFactory` is described in the *mpe_GfxFontFactory* section.

value returned If the call is successful, `mpeos_gfxFontFactoryNew()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_UNKNOWN`
indicates a generic error.

description `mpeos_gfxFontFactoryNew()` should create and initialize a new font factory.

related functions `mpeos_gfxFontFactoryAdd`
`mpeos_gfxFontFactoryDelete`

Screen functions

The surface functions implement the interface for providing information on the current display screen. The screen capabilities contain information such as screen resolution, bit depth, etc. You can also access the internal screen memory, the frame buffer in which the screen pixels are stored. The following screen functions, which are defined in the `mpeos_screen.h`, need to be supported:

`mpeos_gfxCreateDefaultScreen`
creates a default screen.

`mpeos_gfxGetScreen`
gets the address of where the screen information is stored.

`mpeos_gfxScreenResize`
resizes the screen surface.

mpeos_gfxCreateDefaultScreen

creates a default screen.

syntax mpe_Error mpeos_gfxCreateDefaultScreen(void);

parameter(s) none

value returned If the call is successful, mpeos_gfxCreateDefaultScreen() should return MPE_GFX_ERROR_NOERR. Otherwise, it should return one of the following error codes:

MPE_GFX_ERROR_OSERR

indicates an operating system error occurred.

description mpeos_gfxCreateDefaultScreen() should create a screen object with default screen capabilities.

related functions none

mpeos_gfxGetScreen

gets the address of where the screen information is stored.

syntax mpeos_GfxScreen * mpeos_gfxGetScreen(void);

parameter(s) none

value returned If the call is successful, mpeos_gfxGetScreen() should return MPE_GFX_ERROR_NOERR. Otherwise, it should return the following error code:

MPE_GFX_ERROR_INVALID

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description mpeos_gfxGetScreen() should get the address of where the screen information is stored.

related function(s) none

mpeos_gfxScreenResize

resizes the screen surface.

syntax mpe_Error mpeos_gfxScreenResize(int32_t width, int32_t height);

parameter(s) *width* specifies the new width for the screen surface.
height specifies the new height for the screen surface.

value returned If the call is successful, mpeos_gfxScreenResize() should return MPE_GFX_ERROR_NOERR. Otherwise, it should return the following error code:

MPE_GFX_ERROR_INVALID
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description mpeos_gfxScreenResize() should resize the default screen surface to new dimensions.

related function(s) none

Surface functions

The surface functions provide the interface for defining and managing off-screen graphic surfaces. The following surface functions, which are defined in the `mpeos_surface.h`, need to be supported:

- `mpeos_gfxPaletteDelete`
deletes a color palette.
- `mpeos_gfxPaletteGet`
gets new color entries.
- `mpeos_gfxPaletteMatch`
searches for a matching color palette.
- `mpeos_gfxPaletteNew`
creates a new color palette.
- `mpeos_gfxPaletteSet`
creates a copy of an existing color palette.
- `mpeos_gfxSurface`
creates a new screen surface.
- `mpeos_gfxSurfaceCreate`
creates a new off-screen surface using an existing surface.
- `mpeos_gfxSurfaceDelete`
deletes an off-screen surface.
- `mpeos_gfxSurfaceGetInfo`
gets information about a surface.
- `mpeos_gfxSurfaceNew`
creates a new off-screen surface from a surface description.

mpeos_gfxPaletteDelete

deletes a color palette.

syntax mpe_Error mpeos_gfxPaletteDelete(mpe_GfxPalette *palette*);

parameter(s) *palette* specifies the color palette to delete. *palette* is returned from a previous call to mpeos_gfxPaletteNew(). mpe_GfxPalette is described in the *mpe_GfxPalette* section.

value returned If the call is successful, mpeos_gfxPaletteDelete() should return MPE_GFX_ERROR_NOERR. Otherwise, it should return one of the following error codes:

MPE_GFX_ERROR_INVALID

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

description mpeos_gfxPaletteDelete() should delete a previously created color palette.

related function(s) mpeos_gfxPaletteNew

mpeos_gfxPaletteGet

gets new color entries.

syntax `mpe_Error mpeos_gfxPaletteGet(mpe_GfxPalette palette, int nColors, int offset, mpe_GfxColor * colors);`

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>palette</i> | specifies the palette from which to read color entries. <i>palette</i> is returned from a previous call to <code>mpeos_gfxPaletteNew()</code> . <code>mpe_GfxPalette</code> is described in the <i>mpe_GfxPalette</i> section. |
| | <i>nColors</i> | specifies the number of color entries to be copied from the palette into the <i>colors</i> array. |
| | <i>offset</i> | specifies offset of the initial color entry to copy into the <i>colors</i> array. |
| | <i>colors</i> | is an output pointer to an array in which the palette color entries are copied. <code>mpe_GfxColor</code> is described in the <i>mpe_GfxColor</i> section. |

value returned If the call is successful, `mpeos_gfxPaletteGet()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_NOMEM`

indicates there is not enough memory available to perform the specified graphics operation.

description `mpeos_gfxPaletteGet()` should get color entries from the color palette specified by *palette*.

related function(s) `mpeos_gfxPaletteSet`

mpeos_gfxPaletteMatch

searches for a matching color palette.

syntax `mpe_Error mpeos_gfxPaletteMatch(
 mpe_GfxPalette palette,
 mpe_GfxColor color,
 int * index);`

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>palette</i> | specifies the color palette to query. <i>palette</i> is returned from a previous call to <code>mpeos_gfxPaletteNew()</code> . <code>mpe_GfxPalette</code> is described in the <i>mpe_GfxPalette</i> section. |
| | <i>colors</i> | specifies the color for which a matching index is desired. <code>mpe_GfxColor</code> is described in the <i>mpe_GfxColor</i> section. |
| | <i>index</i> | is an output pointer to the color index is to be written. |

value returned If the call is successful, `mpeos_gfxPaletteMatch()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`
indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR`
indicates an operating system error occurred.

description `mpeos_gfxPaletteMatch()` should query the palette for the closest match for the given color. The match is returned as an index into the color palette.

related function(s) `mpeos_gfxPaletteGet`

mpeos_gfxPaletteNew

creates a new color palette.

syntax `mpe_Error mpeos_gfxPaletteNew(int nColors, mpe_GfxPalette * palette);`

parameter(s) `nColors` specifies the desired number of colors of the new palette.
`palette` is an output pointer to where the new palette handle is stored. `mpe_GfxPalette` is described in the *mpe_GfxPalette* section.

value returned If the call is successful, `mpeos_gfxPaletteNew()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID` indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).
`MPE_GFX_ERROR_OSERR` indicates an operating system error occurred.

description `mpeos_gfxPaletteNew()` should create a new color palette capable of holding the specified number of colors.

related function(s) `mpeos_gfxPaletteDelete`
`mpeos_gfxPaletteSet`

mpeos_gfxPaletteSet

creates a copy of an existing color palette.

syntax `mpe_Error mpeos_gfxPaletteSet(`
 `mpe_GfxPalette palette,`
 `mpe_GfxColor * colors,`
 `int nColors,`
 `int offset);`

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>palette</i> | specifies the palette to update with new color entries. <i>palette</i> is returned from a previous call to <code>mpeos_gfxPaletteNew()</code> . <code>mpe_GfxPalette</code> is described in the <i>mpe_GfxPalette</i> section. |
| | <i>colors</i> | is an input pointer to an array of new color entries. <code>mpe_GfxColor</code> is described in the <i>mpe_GfxColor</i> section. |
| | <i>nColors</i> | specifies the number of color entries in <i>colors</i> to copy into the palette. |
| | <i>offset</i> | specifies the offset of the initial color entry within the palette to start copying. |

value returned If the call is successful, `mpeos_gfxPaletteSet()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxPaletteSet()` should create a new color palette capable of holding the specified number of colors.

related function(s) `mpeos_gfxPaletteGet`

mpeos_gfxSurface

creates a new screen surface.

syntax mpeos_GfxSurface * mpeos_gfxSurface(
 mpe_GfxSurfaceInfo * *desc*,
 mpe_Bool *primary*);

parameter(s) *desc* is an input pointer to the surface description.
mpe_GfxSurfaceInfo is described in the *mpe_GfxSurfaceInfo* section.

primary specifies whether the a screen surface is created. If set to TRUE, a screen surface is created. If FALSE, a surface screen is not created. *mpe_Bool* is described in the *mpe_Bool* section.

value returned If the call is successful, *mpeos_gfxSurface()* should create a new screen surface. Otherwise, *mpeos_gfxSurface()* should return NULL.

description *mpeos_gfxSurface()* should create a new *mpeos_GfxSurface* surface from the *mpe_GfxSurfaceInfo* surface description.

related function(s) none

mpeos_gfxSurfaceCreate

creates a new off-screen surface using an existing surface.

syntax `mpe_Error mpeos_gfxSurfaceCreate(
 mpe_GfxSurface base,
 mpe_GfxSurface * surface);`

parameter(s) `base`

specifies the handle to the original surface to use as a prototype. `mpe_GfxSurface` is described in the *mpe_GfxSurface* section.

`surface`

is an output pointer to the location where the surface handle is stored when the call completes successfully. `mpe_GfxSurface` is described in the *mpe_GfxSurface* section.

value returned

If the call is successful, `mpeos_gfxSurfaceCreate()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description

`mpeos_gfxSurfaceCreate()` should create a new off-screen surface using an existing surface as the prototype.

related function(s)

`mpeos_gfxSurfaceDelete`
`mpeos_gfxSurfaceGetInfo`
`mpeos_gfxSurfaceNew`

mpeos_gfxSurfaceDelete

deletes an off-screen surface.

syntax `mpe_Error mpeos_gfxSurfaceDelete(mpe_GfxSurface surface);`

parameter(s) `surface` specifies the handle to the surface to delete. `surface` is returned by a previous call to `mpeos_gfxSurfaceNew()` or `mpeos_gfxSurfaceCreate()`. `mpe_GfxSurface` is described in the *mpe_GfxSurface* section.

value returned If the call is successful, `mpeos_gfxSurfaceDelete()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID`

indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR`

indicates an operating system error occurred.

description `mpeos_gfxSurfaceDelete()` should delete the off-screen surface specified by `surface`.

After `mpeos_gfxSurfaceDelete()` has been called, the off-screen surface should remain valid until all referencing display contexts are also deleted. The screen surface, as returned by `mpeos_dispGetGfxSurface()`, cannot be deleted. Otherwise, undefined behavior may result.

related function(s) `mpeos_gfxSurfaceCreate`
`mpeos_gfxSurfaceGetInfo`
`mpeos_gfxSurfaceNew`

mpeos_gfxSurfaceGetInfo

gets information about a surface.

syntax `mpe_Error mpeos_gfxSurfaceGetInfo(
 mpe_GfxSurface surface,
 mpe_GfxSurfaceInfo * info);`

parameter(s) *surface* specifies the handle to the surface for which information is requested. *surface* is returned by a previous call to either `mpeos_gfxSurfaceNew()` or `mpeos_gfxSurfaceCreate()`. `mpe_GfxSurface` is described in the *mpe_GfxSurface* section.

info is an output pointer to the location of the surface information structure where the information is stored. `mpe_GfxSurfaceInfo` is described in the *mpe_GfxSurfaceInfo* section.

value returned If the call is successful, `mpeos_gfxSurfaceGetInfo()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID` indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR` indicates an operating system error occurred.

description `mpeos_gfxSurfaceGetInfo()` should return a description of the surface specified by *surface*. This information describes the format of the surface and may provide direct access to the pixel map data for the surface.

The surface information structure includes the following:

- ◆ format
- ◆ bits per pixel
- ◆ dimensions
- ◆ bytes per line
- ◆ pixel data
- ◆ optional alpha data and format (see formats defined by [mpe_GfxColorFormat](#))

In general, the number of supported formats for a particular implementation should be kept to a minimum. Perhaps just one format, matching provided by the screen surface, reducing the need for necessary conversions on bit block transfer operations. Where possible, information such as an exclusive surface format should be advertised via a preprocessor macro as well as via `mpeos_gfxSurfaceGetInfo()`.

related function(s)

- mpeos_gfxSurfaceCreate
- mpeos_gfxSurfaceDelete
- mpeos_gfxSurfaceNew

mpeos_gfxSurfaceNew

creates a new off-screen surface from a surface description.

syntax `mpe_Error mpeos_gfxSurfaceNew(
 mpe_GfxSurfaceInfo * desc,
 mpe_GfxSurface * surface);`

parameter(s) `desc` is an input pointer to the surface description to use to create the new off-screen surface. `mpe_GfxSurfaceInfo` is described in the *mpe_GfxSurfaceInfo* section.

`surface` is an output pointer to the location where the surface handle is stored when the call completes successfully. `mpe_GfxSurface` is described in the *mpe_GfxSurface* section.

value returned If the call is successful, `mpeos_gfxSurfaceNew()` should return `MPE_GFX_ERROR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_GFX_ERROR_INVALID` indicates an invalid parameter (for example, out-of-range values, null pointers, unknown values, etc.).

`MPE_GFX_ERROR_OSERR` indicates an operating system error occurred.

description `mpeos_gfxSurfaceNew()` should create a new off-screen surface from the surface description specified by `desc`. The caller provides the surface description. However, the surface description may also be provided by a call to `mpeos_gfxGetSurfaceInfo()`, which gets the surface description of an existing surface.

related function(s) `mpeos_gfxSurfaceCreate`
`mpeos_gfxSurfaceDelete`
`mpeos_gfxSurfaceGetInfo`

User-interface functions

The user-interface function monitors the system for user input events. The following user-interface event function, which is defined in the `mpeos_uievent.h`, needs to be supported:

`mpeos_gfxWaitNextEvent`
waits for the system to generate a user input event.

mpeos_gfxWaitNextEvent

waits for the system to generate a user input event.

syntax mpe_Error mpeos_gfxWaitNextEvent(
 mpe_GfxEvent * *event*,
 uint32_t *timeout*);

parameter(s) *event* is an output pointer to the caller's event structure to be filled. *mpe_GfxEvent* is described in the *mpe_GfxEvent* section.

timeout specifies the length of time to wait in milliseconds. If set to MPE_GFX_WAIT_INFINITE, *mpeos_gfxWaitNextEvent()* wait indefinitely for a user input event.

value returned If the call is successful, *mpeos_gfxWaitNextEvent()* should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPEETIMEOUT indicates the API has returned due to a timeout condition.

description *mpeos_gfxWaitNextEvent()* should wait for the system to generate a user input event for a specified length of time. If an event is received in the specified time, the caller's *mpe_GfxEvent* structure should be filled with the event data.

related function(s) none

MED Media API

Overview

The media Application Programming Interface (API) provides functions for single- and multi-tuner tuning. Tuning can be done using tuning parameters (frequency, program number, and Quadrature Amplitude Modulation (QAM) mode).

The media API provides functions for Moving Picture Experts Group (MPEG) decoder control. These functions provide the mechanism for decoding and playing audio and video encoded in MPEG transport streams.

The media API also provides functions for setting and retrieving video playback and source boundaries, as well as setting volume and mute control.

Before reading this chapter, you should be familiar with:

- ◆ what the media functions are and how they work
- ◆ how your device drivers cause your hardware tuners to tune
- ◆ how you activate decoding for your platform

After reading this chapter, you should be familiar with the media API within the OCAP 1.0 stack, as implemented by Vidiom Systems, Inc.

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the media functions:

MPE_EINVAL
MPE_SUCCESS

MPE_ENOMEM
mpe_MediaErrorCode

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

MPE_EINVAL

MPE_EINVAL indicates at least one input parameter to the function has an invalid value. MPE_EINVAL is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENOMEM

MPE_ENOMEM indicates the function failed due to insufficient memory resource. MPE_ENOMEM is defined as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_SUCCESS

MPE_SUCCESS indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). MPE_SUCCESS is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

mpe_MediaErrorCode mpe_MediaErrorCode specifies the media-specific error codes.
mpe_MediaErrorCode is defined in `mpeos_media.h` as follows:

```
typedef enum _mpe_MediaErrorCode {
    MPE_ERROR_MEDIA_BAD_TUNING_REQUEST =
        OS_MEDIA_ERROR_BADTUNINGREQUEST,
    MPE_ERROR_MEDIA_INVALID_ID =
        OS_MEDIA_ERROR_INVALIDID,
    MPE_ERROR_MEDIA_INVALID_CHANNEL =
        OS_MEDIA_ERROR_INVALIDCHANNEL,
    MPE_ERROR_MEDIA_INVALID_SOURCEID =
        OS_MEDIA_ERROR_INVALIDSOURCEID,
    MPE_ERROR_MEDIA_INVALID_PLAYER =
        OS_MEDIA_ERROR_INVALIDPLAYER,
    MPE_ERROR_MEDIA_NO_LONGER_AUTHORIZED =
        OS_MEDIA_ERROR_NOLONGERAUTHORIZED,
    MPE_ERROR_MEDIA_AUTHORIZATION_FAILED =
        OS_MEDIA_ERROR_AUTHORIZATIONFAILED,
    MPE_ERROR_MEDIA_API_NOT_IMPLEMENTED =
        OS_MEDIA_ERROR_APINOTIMPLEMENTED,
    MPE_ERROR_MEDIA_API_NOT_SUPPORTED =
        OS_MEDIA_ERROR_APINOTSUPPORTED,
    MPE_ERROR_MEDIA_OS = OS_MEDIA_ERROR_GENERIC,
    MPE_ERROR_MEDIA_STREAM_OPEN = OS_MEDIA_ERROR_STREAMOPEN,
    MPE_ERROR_MEDIA_STREAM_READ = OS_MEDIA_ERROR_STREAMREAD,
    MPE_ERROR_MEDIA_BUFFER_OVERRUN =
        OS_MEDIA_ERROR_BUFFEROVERRUN,
    MPE_ERROR_MEDIA_NO_IDS_AVAILABLE =
        OS_MEDIA_ERROR_NOIDSAVAILABLE,
    MPE_ERROR_MEDIA_PRIORITY_LEVEL =
        OS_MEDIA_ERROR_PRIORITYLEVEL,
    MPE_ERROR_MEDIA_RESOURCE_NOT_ACTIVE =
        OS_MEDIA_ERROR_RESOURCENOTACTIVE,
    MPE_ERROR_MEDIA_NOT_OWNER = OS_MEDIA_ERROR_NOTOWNER,
    MPE_ERROR_MEDIA_BAD_LINK = OS_MEDIA_ERROR_BADLINK,
    MPE_ERROR_MEDIA_BLACKED_OUT = OS_MEDIA_ERROR_BLACKEDOUT,
    MPE_ERROR_MEDIA_ECM_STREAM = OS_MEDIA_ERROR_ECMSTREAM,
    MPE_ERROR_MEDIA_NOT_TUNED,
    MPE_ERROR_MEDIA_BAD_PATH,
    MPE_ERROR_MEDIA_RESOURCE_BUSY,
    MPE_ERROR_MEDIA_BAD_QUEUE,
    MPE_ERROR_MEDIA_BAD_DEVICE
} mpe_MediaErrorCode;
```

where

MPE_ERROR_MEDIA_BAD_TUNING_REQUEST
specifies the tune request failed.

MPE_ERROR_MEDIA_INVALID_ID
specifies an invalid tuner identifier.

MPE_ERROR_MEDIA_INVALID_CHANNEL
specifies an invalid channel.

MPE_ERROR_MEDIA_INVALID_SOURCEID
specifies an invalid source identifier.

MPE_ERROR_MEDIA_INVALID_PLAYER
specifies an invalid player.

MPE_ERROR_MEDIA_NO_LONGER_AUTHORIZED
specifies a time-out error occurred on the authorization.

MPE_ERROR_MEDIA_AUTHORIZATION_FAILED
specifies the authorization failed.

MPE_ERROR_MEDIA_API_NOT_IMPLEMENTED
specifies the API is not implemented.

MPE_ERROR_MEDIA_API_NOT_SUPPORTED
specifies the API is not supported by the target platform.

MPE_ERROR_MEDIA_OS
specifies an error occurred in the operating system.

MPE_ERROR_MEDIA_STREAM_OPEN
specifies the stream is open.

MPE_ERROR_MEDIA_STREAM_READ
specifies the is being read.

MPE_ERROR_MEDIA_BUFFER_OVERRUN
specifies the buffer is full.

MPE_ERROR_MEDIA_NO_IDS_AVAILABLE
specifies the identifiers are not available.

MPE_ERROR_MEDIA_PRIORITY_LEVEL
specifies the priority level is not acceptable.

MPE_ERROR_MEDIA_RESOURCE_NOT_ACTIVE
specifies the resource is not active.

MPE_ERROR_MEDIA_NOT_OWNER
specifies an invalid owner.

MPE_ERROR_MEDIA_BAD_LINK
specifies a failed link.

MPE_ERROR_MEDIA_BLACKED_OUT
specifies a black out.

MPE_ERROR_MEDIA_ECM_STREAM
specifies an Entitlement Control Messages (ECM) stream.

MPE_ERROR_MEDIA_NOT_TUNED
specifies the tuner is unable to tune.

MPE_ERROR_MEDIA_BAD_PATH
specifies an invalid path.

MPE_ERROR_MEDIA_RESOURCE_BUSY
specifies the resource is busy (for example, a tuner or a decoder).

MPE_ERROR_MEDIA_BAD_QUEUE
specifies an invalid queue.

MPE_ERROR_MEDIA_BAD_DEVICE
specifies an invalid device.

Data types and structures

The following data types and structures, which are defined in `mpeos_disp.h`, `mpe_error.h`, `mpeos_event.h`, `mpeos_gfx.h`, `mpeos_si.h`, `mpe_types.h`, and `mpeos_media.h`, are used by the media functions:

| | |
|---|---|
| <code>mpe_Bool</code> | <code>mpe_DispDevice</code> |
| <code>mpe_Error</code> | <code>mpe_EventQueue</code> |
| <code>mpe_GfxDimensions</code> | |
| <code>mpe_MediaActiveFormatDescription</code> | |
| <code>mpe_MediaAspectRatio</code> | <code>mpe_MediaDecodeRequestParams</code> |
| <code>mpe_MediaDecodeSession</code> | <code>mpe_MediaErrorCode</code> |
| <code>mpe_MediaRectangle</code> | <code>mpe_MediaTuneParams</code> |
| <code>mpe_MediaTuneRequestParams</code> | <code>mpe_MediaTuneType</code> |
| <code>mpe_SiElemStreamType</code> | <code>mpe_SiModulationMode</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_DispDevice`

`mpe_DispDevice` is a pointer to a device handle. It serves as an abstraction for a screen device belonging to a single screen. The screen is defined as video, graphics, or background type. `mpe_DispDevice` is defined in `mpeos_disp.h` as follows:

```
typedef struct { int unused1; } * mpe_DispDevice;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

`mpe_GfxDimensions`

`mpe_GfxDimensions` specifies the dimensions of the graphics element. `mpe_GfxDimensions` is defined in `mpeos_gfx.h` as follows:

```
typedef struct mpe_GfxDimensions {
    int32 width;
    int32 height;
} mpe_GfxDimensions;
```

where

`width` indicates the width in pixels of the graphic element.

`height` indicates the height in pixels of the graphic element.

mpe_MediaActiveFormatDescription

`mpe_MediaActiveFormatDescription` specifies the MPEG Active Format Description (AFD) of the video being presented.

`mpe_MediaActiveFormatDescription` is defined in `mpeos_media.h` as follows:

```
typedef enum {
    MPE_AFD_NOT_PRESENT = -1,
    MPE_AFD_16_9_TOP = 2,
    MPE_AFD_14_9_TOP,
    MPE_AFD_GT_16_9,
    MPE_AFD_SAME = 8,
    MPE_AFD_4_3,
    MPE_AFD_16_9,
    MPE_AFD_14_9,
    MPE_AFD_4_3_SP_14_9 = 13,
    MPE_AFD_16_9_SP_14_9,
    MPE_AFD_16_9_SP_4_3
} mpe_MediaActiveFormatDescription;
```

where

- MPE_AFD_NOT_PRESENT indicates the format constant is not present.
- MPE_AFD_16_9_TOP indicates the format is 16:9 (top).
- MPE_AFD_14_9_TOP indicates the format is 14:9 (top).
- MPE_AFD_GT_16_9 indicates the format is greater than 16:9 (center).
- MPE_AFD_SAME indicates the format is the same as the coded frame.
- MPE_AFD_4_3 indicates the format is 4:3 (center).
- MPE_AFD_16_9 indicates the format is 16:9 (center).
- MPE_AFD_14_9 indicates the format is 14:9 (center).
- MPE_AFD_4_3_SP_14_9 indicates the format is 4:3 (with shoot and protect 14:9 center).
- MPE_AFD_16_9_SP_14_9 indicates the format is 16:9 (with shoot and protect 14:9 center).
- MPE_AFD_16_9_SP_4_3 indicates the format is 16:9 (with shoot and protect 4:3 center).

mpe_MediaAspectRatio

`mpe_MediaAspectRatio` specifies the ratio of the video being presented.
`mpe_MediaAspectRatio` is defined in `mpeos_media.h` as follows:

```
typedef enum {
    MPE_ASPECT_RATIO_UNKNOWN = -1,
    MPE_ASPECT_RATIO_4_3 = 2,
    MPE_ASPECT_RATIO_16_9,
    MPE_ASPECT_RATIO_2_21_1
} mpe_MediaAspectRatio;
```

where

`MPE_ASPECT_RATIO_UNKNOWN`
indicates an unknown ratio.

`MPE_ASPECT_RATIO_4_3`
indicates an 4:3 ratio.

`MPE_ASPECT_RATIO_16_9`
indicates an 16:9 ratio.

`MPE_ASPECT_RATIO_2_21_1`
indicates an 2.21:1 ratio.

mpe_MediaDecodeRequestParams

`mpe_MediaDecodeRequestParams` specifies the request parameters for the decoder. `mpe_MediaDecodeRequestParams` is defined in `mpeos_media.h` as follows:

```
typedef struct _mpe_MediaDecodeRequestParams {
    uint32_t tunerId;
    mpe_DispDevice videoDevice;
    uint32_t numPids;
    mpe_MediaPID *pids;
    uint32_t pcrPid;
} mpe_MediaDecodeRequestParams;
```

where

- `tunerId` specifies the tuner identifier.
- `videoDevice` specifies a pointer to a device handle. `mpe_Dispatcher` is described in the *mpe_Dispatcher* section.
- `numPids` specifies the number of Packet Identifications (PID).
- `pids` is a pointer to the array of PID. `mpe_MediaPID` is described in the *mpe_MediaPid* section.
- `pcrPid` indicates the PID value on which the Program Clock Reference (PCR) values are specified by the program being decoded.

mpe_MediaDecodeSession

`mpe_MediaDecodeSession` identifies the media decode session handle. `mpe_MediaDecodeSession` is defined in `mpeos_media.h` as follows:

```
typedef struct _mpe_MediaDecodeSessionH { int unused1; }
*mpe_MediaDecodeSession;
```

mpe_MediaDripFeedRequestParams

`mpe_MediaDripFeedRequestParams` defines the media decode session handle. `mpe_MediaDripFeedRequestParams` is defined in `mpeos_media.h` as follows:

```
typedef struct _mpe_MediaDripFeedRequestParams {
    mpe_Dispatcher videoDevice;
} mpe_MediaDripFeedRequestParams;
```

where

- `videoDevice` specifies the device handle. `mpe_Dispatcher` is described in the *mpe_Dispatcher* section.

mpe_MediaPid

`mpe_MediaPID` specifies the type of elementary stream_type field in the Program Map Table (PMT) and the PID value. `mpe_MediaPID` is defined in `mpeos_media.h` as follows:

```
typedef struct _mpe_MediaPID {
    mpe_SiElemStreamType pidType;
    uint32_t pid;
} mpe_MediaPID;
```

where

`pidType` specifies the PID type (for example, audio, video, closed captioning, etc.) for the elementary stream_type field in the PMT. `mpe_SiElemSteamType` is described in the *mpe_SiElemStreamType* section.

`pid` indicates the PID value.

mpe_MediaPositioningCapabilities

`mpe_MediaPositioningCapabilities` specifies how the video can be positioned on the screen. Currently, the values for this are -1, 0, 1, 3, or 4. These values correspond to the `POS_CAP_*` constant values defined in the OCAP Specification (OC-SP-OCAPI.0-I16).

`mpe_MediaPositioningCapabilities` is defined in `mpeos_media.h` as follows:

```
typedef enum _mpe_MediaPositioningCapabilities {
    MPE_POS_CAP_OTHER = -1,
    MPE_POS_CAP_FULL = 0,
    MPE_POS_CAP_FULL_IF_ENTIRE_VIDEO_ON_SCREEN,
    MPE_POS_CAP_FULL_EVEN_LINES = 3,
    MPE_POS_CAP_FULL_EVEN_LINES_IF_ENTIRE_VIDEO_ON_SCREEN
} mpe_MediaPositioningCapabilities;
```

where

`MPE_POS_CAP_OTHER`

specifies the video positioning capability cannot be expressed by another `POS_CAP_*` constant.

`MPE_POS_CAP_FULL`

specifies the video can be positioned anywhere on the screen, even if a part of the video is off screen.

`MPE_POS_CAP_FULL_IF_ENTIRE_VIDEO_ON_SCREEN`

specifies the video can be positioned anywhere on screen as long as all the video is on screen.

`MPE_POS_CAP_FULL_EVEN_LINES`

specifies the video can be positioned anywhere on the screen, even if a part of the video is off screen, with the restriction that the field order is respected. This implies that interlaced video can be positioned on even lines only (when counting from 0).

MPE_POS_CAP_FULL_EVEN_LINES_IF_ENTIRE_VIDEO_ON_SCREEN
 specifies the video can be positioned anywhere on screen as long as all the video is on screen, with the restriction that the field order is respected. This implies that interlaced video can be positioned on even lines only (when counting from 0).

mpe_MediaRectangle `mpe_MediaRectangle` defines the origin and dimensions of a rectangle. `mpe_MediaRectangle` is defined in `mpeos_media.h` as follows:

```
typedef mpe_DispScreenArea mpe_MediaRectangle;
```

mpe_MediaTuneParams

`mpe_MediaTuneParams` specifies the tuning parameters. `mpe_MediaTuneParams` is defined in `mpeos_media.h` as follows:

```
typedef struct _mpe_MediaTuneParams {
    mpe_MediaTuneType tuneType;
    uint32_t frequency;
    uint32_t programNumber;
    mpe_SiModulationMode qamMode;
} mpe_MediaTuneParams;
```

where

`tuneType` identifies the type of tune. `mpe_MediaTuneType` is described in the *mpe_MediaTuneType* section.

`frequency` identifies the frequency.

`programNumber` identifies the program number.

`qamMode` identifies QAM level. `mpe_SiModulationMode` is described in the *mpe_SiModulationMode* section.

mpe_MediaTuneRequestParams

`mpe_MediaTuneRequestParams` specifies the requested parameters for the tuner. `mpe_MediaTuneRequestParams` is defined in `mpeos_media.h` as follows:

```
typedef struct _mpe_MediaTuneRequestParams {
    uint32_t tunerId;
    uint32_t tsId;
    mpe_MediaTuneParams tuneParams;
} mpe_MediaTuneRequestParams;
```

where

`tunerId` identifies the tuner identifier.

- ◆ **NOTE:** `tunerId` 0 is reserved for Out-Of-Band (OOB). Tuner identifiers (`tunerId`) for in-band tuners start at 1 and increment depending on the set-top box capability.

- tsId identifies the transport stream identifier.
- tuneParams identifies the tuning parameters, which includes frequency, program number, and QAM mode. `mpe_MediaTuneParams` is described in the *mpe_MediaTuneParams* section.

mpe_MediaTuneType `mpe_MediaTuneType` identifies the type of tune. `mpe_MediaTuneType` is defined in `mpeos_media.h` as follows:

```
typedef enum _mpe_MediaTuneType {
    MPE_MEDIA_TUNE_BY_TUNING_PARAMS,
    MPE_MEDIA_TUNE_BY_UNKNOWN
} mpe_MediaTuneType;
```

where

`MPE_MEDIA_TUNE_BY_TUNING_PARAMS`
specifies the tuner uses tuning parameters to tune.

`MPE_MEDIA_TUNE_BY_UNKNOWN`
specifies the tuner uses an unknown source to tune.

mpe_SiElemStreamType

`mpe_SiElemStreamType` defines valid values for the elementary stream_type field in the PMT. On some platforms, elementary steam type definitions can be mapped to PID types directly. Otherwise, they need to be manually mapped to what the operating system requires.

`mpe_SiElemStreamType` is defined in `mpeos_si.h` as follows:

```
typedef enum mpe_SiElemStreamType {
    MPE_SI_ELEM_MPEG_1_VIDEO = 0x01,
    MPE_SI_ELEM_MPEG_2_VIDEO = 0x02,
    MPE_SI_ELEM_MPEG_1_AUDIO = 0x03,
    MPE_SI_ELEM_MPEG_2_AUDIO = 0x04,
    MPE_SI_ELEM_MPEG_PRIVATE_SECTION = 0x05,
    MPE_SI_ELEM_MPEG_PRIVATE_DATA = 0x06,
    MPE_SI_ELEM_MHEG = 0x07,
    MPE_SI_ELEM_DSM_CC = 0x08,
    MPE_SI_ELEM_H_222 = 0x09,
    MPE_SI_ELEM_DSM_CC_MPE = 0x0A,
    MPE_SI_ELEM_DSM_CC_UN = 0x0B,
    MPE_SI_ELEM_DSM_CC_STREAM_DESCRIPTORS = 0x0C,
    MPE_SI_ELEM_DSM_CC_SECTIONS = 0x0D,
    MPE_SI_ELEM_AUXILIARY = 0x0E,
    MPE_SI_ELEM_VIDEO_DCII = 0x80,
    MPE_SI_ELEM_ATSC_AUDIO = 0x81,
    MPE_SI_ELEM_STD_SUBTITLE = 0x82,
    MPE_SI_ELEM_ISOCHRONOUS_DATA = 0x83,
    MPE_SI_ELEM_ASYNCHRONOUS_DATA = 0x84
} mpe_SiElemStreamType;
```

where

- MPE_SI_ELEM_MPEG_1_VIDEO
 - specifies Moving Picture Experts Group - video Compact Disk (CD) (MPEG-1) video and is assigned 0x01.
- MPE_SI_ELEM_MPEG_2_VIDEO
 - specifies Moving Picture Experts Group - broadcast signals (MPEG-2) video and is assigned 0x02.
- MPE_SI_ELEM_MPEG_1_AUDIO
 - specifies MPEG-1 audio and is assigned 0x03.
- MPE_SI_ELEM_MPEG_2_AUDIO
 - specifies MPEG-2 audio and is assigned 0x04.
- MPE_SI_ELEM_MPEG_PRIVATE_SECTION
 - specifies MPEG private section and is assigned 0x05.
- MPE_SI_ELEM_MPEG_PRIVATE_DATA
 - specifies MPEG private data and is assigned 0x06.
- MPE_SI_ELEM_MHEG
 - specifies multimedia and hypermedia information coding Expert Group (MHEG) and is assigned 0x07.
- MPE_SI_ELEM_DSM_CC
 - specifies Digital Storage Media - Command and Control (DSM-CC) and is assigned 0x08.
- MPE_SI_ELEM_H_222
 - specifies the international standard, Recommendation H.222.0 and is assigned 0x09.
- MPE_SI_ELEM_DSM_CC_MPE
 - specifies DSM-CC MPE and is assigned 0x0A.
- MPE_SI_ELEM_DSM_CC_UN
 - specifies Digital Storage Media - Command and Control User-to-Network (DSM-CC UN) and is assigned 0x0B.
- MPE_SI_ELEM_DSM_CC_STREAM_DESCRIPTORS
 - specifies DSM-CC stream descriptors and is assigned 0x0C.
- MPE_SI_ELEM_DSM_CC_SECTIONS
 - specifies DSM-CC sections and is assigned 0x0D.
- MPE_SI_ELEM_AUXILIARY
 - specifies auxiliary and is assigned 0x0E.
- MPE_SI_ELEM_VIDEO_DCII
 - specifies video Digicipher II (DCII) and is assigned 0x80.
- MPE_SI_ELEM_ATSC_AUDIO
 - specifies Advanced Television Systems Committee (ATSC) audio and is assigned 0x81.
- MPE_SI_ELEM_STD_SUBTITLE
 - specifies standard subtitle and is assigned 0x82.

MPE_SI_ELEM_ISOCHRONOUS_DATA
specifies synchronous data and is assigned 0x83.

MPE_SI_ELEM_ASYNCHRONOUS_DATA
specifies asynchronous data and is assigned 0x84.

mpe_SiModulationMode

`mpe_SiModulationMode` defines the modulation mode used by the head-end for tuning parameters. `mpe_SiModulationMode` is defined in `mpeos_si.h` as follows:

```
typedef enum mpe_SiModulationMode {
    MPE_SI_MODULATION_UNKNOWN=0,
    MPE_SI_MODULATION_QPSK,
    MPE_SI_MODULATION_BPSK,
    MPE_SI_MODULATION_OQPSK,
    MPE_SI_MODULATION_VSB8,
    MPE_SI_MODULATION_VSB16,
    MPE_SI_MODULATION_QAM16,
    MPE_SI_MODULATION_QAM32,
    MPE_SI_MODULATION_QAM64,
    MPE_SI_MODULATION_QAM80,
    MPE_SI_MODULATION_QAM96,
    MPE_SI_MODULATION_QAM112,
    MPE_SI_MODULATION_QAM128,
    MPE_SI_MODULATION_QAM160,
    MPE_SI_MODULATION_QAM192,
    MPE_SI_MODULATION_QAM224,
    MPE_SI_MODULATION_QAM256,
    MPE_SI_MODULATION_QAM320,
    MPE_SI_MODULATION_QAM384,
    MPE_SI_MODULATION_QAM448,
    MPE_SI_MODULATION_QAM512,
    MPE_SI_MODULATION_QAM640,
    MPE_SI_MODULATION_QAM768,
    MPE_SI_MODULATION_QAM896,
    MPE_SI_MODULATION_QAM1024,
    MPE_SI_MODULATION_QAM_NTSC = 255
} mpe_SiModulationMode;
```

where

MPE_SI_MODULATION_UNKNOWN
specifies an unknown modulation mode.

MPE_SI_MODULATION_QPSK
specifies Quadrature Phase-Shift Keying (QPSK) modulation mode.

MPE_SI_MODULATION_BPSK
specifies Binary Phase-Shift Keying (BPSK) modulation mode.

MPE_SI_MODULATION_OQPSK
specifies Offset Quadrature Phase-Shift Keying (OQPSK) modulation mode.

MPE_SI_MODULATION_VSB8
specifies 8-level Vestigial Sideband (VSB8) modulation mode.

MPE_SI_MODULATION_VSB16
specifies 16-level Vestigial Sideband (VSB16) modulation mode.

MPE_SI_MODULATION_QAM16
specifies Quadrature Amplitude Modulation (QAM) QAM16 modulation mode.

MPE_SI_MODULATION_QAM32
specifies QAM32 modulation mode.

MPE_SI_MODULATION_QAM64
specifies QAM64 modulation mode.

MPE_SI_MODULATION_QAM80
specifies QAM80 modulation mode.

MPE_SI_MODULATION_QAM96
specifies QAM96 modulation mode.

MPE_SI_MODULATION_QAM112
specifies QAM112 modulation mode.

MPE_SI_MODULATION_QAM128
specifies QAM128 modulation mode.

MPE_SI_MODULATION_QAM160
specifies QAM160 modulation mode.

MPE_SI_MODULATION_QAM192
specifies QAM192 modulation mode.

MPE_SI_MODULATION_QAM224
specifies QAM224 modulation mode.

MPE_SI_MODULATION_QAM256
specifies QAM256 modulation mode.

MPE_SI_MODULATION_QAM320
specifies QAM320 modulation mode.

MPE_SI_MODULATION_QAM384
specifies QAM384 modulation mode.

MPE_SI_MODULATION_QAM448
specifies QAM448 modulation mode.

MPE_SI_MODULATION_QAM512
specifies QAM512 modulation mode.

MPE_SI_MODULATION_QAM640
specifies QAM640 modulation mode.

MPE_SI_MODULATION_QAM768

specifies QAM768 modulation mode.

MPE_SI_MODULATION_QAM896

specifies QAM896 modulation mode.

MPE_SI_MODULATION_QAM1024

specifies QAM1024 modulation mode.

MPE_SI_MODULATION_QAM_NTSC

specifies National Television System(s) Committee
(NTSC) (or analog) modulation mode and is assigned 255.

Events

The media functions need events, which are defined in `mpeos_media.h`, in the following categories:

- ◆ *Common*
- ◆ *Decoding*
- ◆ *Drip feed*
- ◆ *Tuning*

Common

Common events are used by both the decoder and the tuner. The following common events are used by the media functions:

`MPE_EVENT_SHUTDOWN`

`MPE_EVENT_SHUTDOWN`

`MPE_EVENT_SHUTDOWN` is sent to the decoder or tuner during clean-up. `MPE_EVENT_SHUTDOWN` should be sent when no further events are sent to the queue. `MPE_EVENT_SHUTDOWN` is defined as follows:

```
#define MPE_EVENT_SHUTDOWN 0x8
```

Decoding

Decoding events are the direct result of a request to decode specific program media streams via `mpeos_mediaDecode()`. These events are expected to be sent as close to the time they actually occur as possible. The following decoding events are used by the media functions:

| | |
|--|--|
| <code>MPE_ACTIVE_FORMAT_CHANGED</code> | <code>MPE_ASPECT_RATIO_CHANGED</code> |
| <code>MPE_FAILURE_CA_DENIED</code> | <code>MPE_FAILURE_NO_DATA</code> |
| <code>MPE_FAILURE_UNKNOWN</code> | <code>MPE_STREAM_CA_DENIED</code> |
| <code>MPE_STREAM_DIALOG_PAYMENT</code> | <code>MPE_STREAM_DIALOG_RATING</code> |
| <code>MPE_STREAM_DIALOG_TECHNICAL</code> | <code>MPE_STREAM_HW_UNAVAILABLE</code> |
| <code>MPE_STREAM_NO_DATA</code> | <code>MPE_STREAM_RETURNED</code> |

`MPE_ACTIVE_FORMAT_CHANGED`

`MPE_ACTIVE_FORMAT_CHANGED` is asynchronously sent, in response to Active Format Descriptor (AFD) changes from the hardware decoder, to applications while the video is being decoded. AFD describes the input video format, either letter boxed or pillar boxed (typically 4:3 for standard definition and 16:9 for high definition). `MPE_ACTIVE_FORMAT_CHANGED` is defined as follows:

```
#define MPE_ACTIVE_FORMAT_CHANGED 0x15
```

`MPE_ASPECT_RATIO_CHANGED`

`MPE_ASPECT_RATIO_CHANGED` is asynchronously sent, in response to aspect ratio changes from the hardware decoder, to applications while the video is being decoded. `MPE_ASPECT_RATIO_CHANGED` is defined as follows:

```
#define MPE_ASPECT_RATIO_CHANGED 0x16
```

MPE_CONTENT_PRESENTING

MPE_CONTENT_PRESENTING is asynchronously sent to applications when a decode request fails because it was denied by conditional access.

MPE_CONTENT_PRESENTING is defined as follows:

```
#define MPE_CONTENT_PRESENTING 0x05
```

MPE_DFC_CHANGED

MPE_DFC_CHANGED is sent to the decoder queue if the implementation changes the current Decoder Format Conversion (DFC).

MPE_DFC_CHANGED is not optional for implementation that can control the DFC filter. MPE_DFC_CHANGED is defined as follows:

```
#define MPE_DFC_CHANGED 0x17
```

MPE_FAILURE_CA_DENIED

MPE_FAILURE_CA_DENIED is asynchronously sent, in response to denied conditional access failure from the hardware decoder, to applications while the video is being decoded. MPE_FAILURE_CA_DENIED is defined as follows:

```
#define MPE_FAILURE_CA_DENIED 0x0A
```

MPE_FAILURE_NO_DATA

MPE_FAILURE_NO_DATA is sent to the decoder queue when the request fails because data is no longer being delivered (for example, the cable transport has been unplugged). MPE_FAILURE_NO_DATA is defined as follows:

```
#define MPE_FAILURE_NO_DATA 0x0F
```

MPE_FAILURE_UNKNOWN

MPE_FAILURE_UNKNOWN is asynchronously sent, in response to an unknown failure from the hardware decoder, to applications while video is being decoded. MPE_FAILURE_UNKNOWN is defined as follows:

```
#define MPE_FAILURE_UNKNOWN 0x06
```

MPE_STREAM_CA_DENIED

MPE_STREAM_CA_DENIED indicates a stream is not decoding because it was denied by conditional access. MPE_STREAM_CA_DENIED is defined as follows:

```
#define MPE_STREAM_DENIED 0x0B
```

MPE_STREAM_CA_UNKNOWN

MPE_STREAM_CA_UNKNOWN is asynchronously sent back to the application when a stream is not decoding because the cipher is unknown. Unlike other events, it is not passed into or generated from any function call. MPE_STREAM_CA_UNKNOWN is defined as follows:

```
#define MPE_STREAM_UNKNOWN 0x10
```

MPE_STREAM_DIALOG_PAYMENT

MPE_STREAM_DIALOG_PAYMENT indicates a stream is not decoding because a payment dialog is required. MPE_STREAM_DIALOG_PAYMENT is defined as follows:

```
#define MPE_STREAM_DIALOG_PAYMENT 0x11
```

MPE_STREAM_DIALOG_RATING

MPE_STREAM_DIALOG_RATING indicates a stream is not decoding because a rating dialog is required. MPE_STREAM_DIALOG_RATING is defined as follows:

```
#define MPE_STREAM_DIALOG_RATING 0x13
```

MPE_STREAM_DIALOG_TECHNICAL

MPE_STREAM_DIALOG_TECHNICAL indicates a stream is not decoding because a technical dialog is required. MPE_STREAM_DIALOG_TECHNICAL is defined as follows:

```
#define MPE_STREAM_DIALOG_TECHNICAL 0x12
```

MPE_STREAM_HW_UNAVAILABLE

MPE_STREAM_HW_UNAVAILABLE indicates a stream is not decoding because a hardware resource is unavailable. MPE_STREAM_HW_UNAVAILABLE is defined as follows:

```
#define MPE_STREAM_HW_UNAVAILABLE 0x14
```

MPE_STREAM_NO_DATA

MPE_STREAM_NO_DATA indicates a stream is not decoding because it is not present. MPE_STREAM_NO_DATA is defined as follows:

```
#define MPE_STREAM_NO_DATA 0x0D
```

MPE_STREAM_RETURNED

MPE_STREAM_RETURNED indicates a requested stream is now being decoded. MPE_STREAM_RETURNED is defined as follows:

```
#define MPE_STREAM_RETURNED 0x0C
```

Drip feed

The drip-feed event is the direct result of a request to decode specific still frame via `mpeos_mediaTune()`. These events are expected to be sent as close to the time they actually occur as possible. The following decoding drip-free event is used by the media functions:

```
MPE_STILL_FRAME_DECODED
```

MPE_STILL_FRAME_DECODED

MPE_STILL_FRAME_DECODED indicates a drip-feed frame was decoded. MPE_STILL_FRAME_DECODED is defined as follows:

```
#define MPE_STILL_FRAME_DECODED 0x07
```

Tuning

Tuning events are the direct result of a request to tune to some specified parameters via `mpeos_mediaTune()`. These events are expected to be sent as close to the time they actually occur as possible. The following tuning events are used by the media functions:

| | |
|-----------------------------|--------------------------------|
| <code>MPE_TUNE_ABORT</code> | <code>MPE_TUNE_COMPLETE</code> |
| <code>MPE_TUNE_FAIL</code> | <code>MPE_TUNE_STARTED</code> |

MPE_TUNE_ABORT `MPE_TUNE_ABORT` is sent for the following reasons:

- ◆ tuning parameters change (event sent to the queue associated with the original tune request)
- ◆ an asynchronous failure occurred during or after tuning (event sent to the queue associated with the most recent tune request)

`MPE_TUNE_ABORT` is defined as follows:

```
#define MPE_TUNE_ABORT 0x4
```

MPE_TUNE_COMPLETE

`MPE_TUNE_COMPLETE` is sent once the tune is complete (for example, the output signal from the tuner is stable). `MPE_TUNE_COMPLETE` is defined in as follows:

```
#define MPE_TUNE_COMPLETE 0x2
```

MPE_TUNE_FAIL `MPE_TUNE_FAIL` is sent as an alternative to `MPE_TUNE_COMPLETE` and thus

should be sent if the tuner was unable to tune for any reason.

`MPE_TUNE_FAIL` is defined as follows:

```
#define MPE_TUNE_FAIL 0x3
```

MPE_TUNE_STARTED

`MPE_TUNE_STARTED` is sent when a tune has been requested and that request is being processed. `MPE_TUNE_STARTED` is defined as follows:

```
#define MPE_TUNE_STARTED 0x9
```

Supported functions

The media functions are used to initialize and clean up after the media API. The following general functions need to be supported:

- `mpeos_mediaCheckBounds`
verifies the desired bounds are supported.
- `mpeos_mediaDecode`
starts presenting the specified media.
- `mpeos_mediaDripFeedRenderFrame`
renders a single MPEG-2 video frame synchronously.
- `mpeos_mediaDripFeedStart`
initializes a drip-feed decode session.
- `mpeos_mediaDripFeedStop`
stops a drip-feed session.
- `mpeos_mediaFreeze`
freezes the media while presenting a single media frame.
- `mpeos_mediaFrequencyToTuner`
identifies the tuner using the specified frequency.
- `mpeos_mediaGetAFD`
gets the current AFD for the decoder.
- `mpeos_mediaGetAspectRatio`
gets the aspect ratio for the video.
- `mpeos_mediaGetBounds`
gets the display dimensions for the target display device.
- `mpeos_mediaGetInputVideoSize`
gets the video size.
- `mpeos_mediaGetScaling`
gets the scaling capabilities of the specified decoder.
- `mpeos_mediaGetTunerInfo`
gets the current tuner information.
- `mpeos_mediaInit`
initializes the media API.
- `mpeos_mediaRegisterQueueForTuneEvents`
registers an event queue.
- `mpeos_mediaResume`
resumes the media decoding.
- `mpeos_mediaSetBounds`
sets the bounds of a display device.
- `mpeos_mediaShutdown`
sends shut down events to the tuner and decoder.
- `mpeos_mediaStop`
stops presenting the media.

mpeos_mediaSwapDecoders
swaps the video source feeds to the two specified decoders.

mpeos_mediaTune
starts the tune operation.

mpeos_mediaUnregisterQueue
unregisters an event queue.

mpeos_mediaCheckBounds

verifies the desired bounds are supported.

syntax `mpe_Error mpeos_mediaCheckBounds(mpe_DispDevice videoDevice, mpe_MediaRectangle * desiredSrc, mpe_MediaRectangle * desiredDst, mpe_MediaRectangle * actualSrc, mpe_MediaRectangle * actualDst);`

| | | |
|----------------------------|---|--|
| parameter(s) | <i>videoDevice</i> | specifies the display device. <code>mpe_DispDevice</code> is described in the <i>mpe_DispDevice</i> section. |
| | <i>desiredSrc</i> | is an input pointer to origin and dimensions of a source rectangle. <code>mpe_MediaRectangle</code> is described in the <i>mpe_MediaRectangle</i> section. |
| | <i>desiredDst</i> | is an input pointer to origin and dimensions of a destination rectangle. <code>mpe_MediaRectangle</code> is described in the <i>mpe_MediaRectangle</i> section. |
| | <i>actualSrc</i> | is an output pointer to origin and dimensions of a destination rectangle. <code>mpe_MediaRectangle</code> is described in the <i>mpe_MediaRectangle</i> section. |
| | <i>actualDst</i> | is an output pointer to origin and dimensions of a destination rectangle. <code>mpe_MediaRectangle</code> is described in the <i>mpe_MediaRectangle</i> section. |
| value returned | If the call is successful, <code>mpeos_mediaCheckBounds()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return the following error code: | |
| | <code>MPE_ERROR_MEDIA_OS</code> specifies an operating system error. | |
| description | <code>mpeos_mediaCheckBounds()</code> should check the desired source and destination bounds for the target display device against platform supported bounds. If the desired bounds are not supported, the closest supported approximation is returned via the actual rectangles. | |
| related function(s) | <code>mpeos_mediaGetBounds</code> <code>mpeos_mediaSetBounds</code> | |

mpeos_mediaDecode

starts presenting the specified media.

syntax

```
mpe_Error mpeos_mediaDecode(
    mpe_MediaDecodeRequestParams * decodeRequest,
    mpe_EventQueue queueId ,
    void * act,
    mpe_MediaDecodeSession * session ) ;
```

parameter(s) *decodeRequest* is an input pointer to the decode request parameters.

mpe_MediaDecodeRequestParams is described in the *mpe_MediaDecodeRequestParams* section.

queueId identifies the queue in which to send the decode events associated with this decode request. *mpe_EventQueue* is described in the *mpe_EventQueue* section.

If one or more of the requested streams cannot be presented, the reason for the stream failure is indicated by sending a stream event for each such stream followed by a MPE_CONTENT_PRESENTING event. *mpeos_mediaDecode()* continues to monitor the failed streams until either *mpeos_mediaStop()* or *mpeos_mediaDecode()* is called.

- ◆ **NOTE:** If an error is returned, no future event are delivered.

Currently, the following events are supported:

MPE_EVENT_SHUTDOWN

indicates no further events can be sent to the queue.

MPE_ACTIVE_FORMAT_CHANGED

indicates the AFD type of input video, either letter boxed or pillar boxed.

MPE_ASPECT_RATIO_CHANGED

indicates the aspect ratio changed.

MPE_DFC_CHANGED

indicates the current DFC changed.

During a full decoder stop all decoding stops and does not restart until another call to *mpeos_mediaDecode()* is made. A full decoder stop is indicated by sending one of the following events:

MPE_FAILURE_CA_DENIED

indicates a stream is not decoding because it was denied by conditional access.

MPE_FAILURE_NO_DATA

indicates the decode request has failed because data is no longer being delivered (for example, the cable transport has been unplugged).

MPE_FAILURE_UNKNOWN

indicates the decode request has failed due to an unknown reason.

act

is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous events are sent to the event queue specified in *queueId* (via `mpeos_eventQueueSend()`), the event should pass this *act* pointer in the *optionalEventData2* parameter.

session

is an output pointer to store the new decode session. If the function fails, the pointer is set to point to NULL.
`mpe_MediaDecodeSession` is described in the `mpe_MediaDecodeSession` section.

value returned

If the call is successful, `mpeos_mediaDecode()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE_ERROR_MEDIA_INVALID_ID`
 specifies an invalid tuner identifier.

`MPE_ERROR_MEDIA_RESOURCE_NOT_ACTIVE`
 specifies the resource is not available.

description

`mpeos_mediaDecode()` should start presenting the media streams specified by the PIDs. A successful call to `mpeos_mediaTune()` must be made before calling `mpeos_mediaDecode()`. Subsequent calls to `mpeos_mediaDecode()` that include PIDs which are already presenting, must continue to present those streams without interruption.

related function(s)

- `mpeos_mediaFreeze`
- `mpeos_mediaResume`
- `mpeos_mediaStop`
- `mpeos_mediaSwapDecoders`
- `mpeos_mediaTune`

mpeos_mediaDripFeedRenderFrame

renders a single MPEG-2 video frame synchronously.

syntax `mpe_Error mpeos_mediaDripFeedRenderFrame(`
`mpe_MediaDecodeSession mediaDecodeSession,`
`uint8_t * buffer,`
`size_t length);`

parameter(s) *mediaDecodeSession*

is an input pointer to the video-drip feed session.
mpe_MediaDecodeSession is described in the
mpe_MediaDecodeSession section.

buffer is an input pointer the byte array containing the MPEG-2 I-Frame or P-Frame.

length specifies the number of bytes in *buffer*.

value returned If the call is successful, `mpeos_mediaDripFeedRenderFrame()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_mediaDripFeedRenderFrame()` should render a single MPEG-2 video frame synchronously. It should receive an I- or P-frames that do not require future frames for decode. `mpeos_mediaDripFeedRenderFrame()` sends the `MPE_STILL_FRAME_DECODED` event to the queue identifier of `mpeos_mediaDripFeedStart()`.

related function(s) `mpeos_mediaDripFeedStart`
`mpeos_mediaDripFeedStop`

mpeos_mediaDripFeedStart

initializes a drip-feed decode session.

syntax

```
mpe_Error mpeos_mediaDripFeedStart(
    mpe_MediaDripFeedRequestParams * dripFeedRequest,
    mpe_EventQueue queueId ,
    void * act,
    mpe_MediaDecodeSession * mediaDecodeSession );
```

parameter(s) *dripFeedRequest*

is an input pointer to the media decode session handle.
mpe_MediaDripFeedRequestParams is described in the
mpe_MediaDripFeedRequestParams section.

queueId

identifies the queue in which to send the events.
mpe_EventQueue is described in the *mpe_EventQueue* section. Currently, the following event is supported:

MPE_STILL_FRAME_DECODED

indicates a drip-feed frame was decoded.

act

is an input pointer to an ACT. When the asynchronous events are sent to the event queue specified in *queueId* (via *mpeos_eventQueueSend()*), the event should pass this *act* pointer in the *optionalEventData2* parameter.

mediaDecodeSession

is an output pointer to the video-drip feed session. If the function fails, the pointer is set to point to NULL.
mpe_MediaDecodeSession is described in the *mpe_MediaDecodeSession* section.

value returned

If the call is successful, *mpeos_mediaDripFeedStart()* should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL

indicates at least one input parameter to the function has an invalid value.

MPE_ERROR_MEDIA_RESOURCE_BUSY

indicates the resource is busy (for example, a tuner or a decoder).

description

mpeos_mediaDripFeedStart() should initialize a drip-feed decode session based on the input parameters. The decode session can be used to later submit frames of data for decode and to stop the decode. For now, other aspects of the drip feed (such as bounds) can be manipulated using the associated video device with other *mpe_media* routines. In the future, the intent is to have all media routines accept a decode session.

related function(s)

mpeos_eventQueueNew

mpeos_mediaDripFeedRenderFrame

mpeos_mediaDripFeedStop

mpeos_mediaDripFeedStop

stops a drip-feed session.

syntax mpe_Error mpeos_mediaDripFeedStop(
 mpe_MediaDecodeSession mediaDecodeSession);

parameter(s) *mediaDecodeSession*

is an input pointer to the video-drip feed session.

mediaDecodeSession must have been returned by a previous call to `mpeos_mediaDripFeedStart()`.

`mpe_MediaDecodeSession` is described in the `mpe_MediaDecodeSession` section.

value returned If the call is successful, `mpeos_mediaDripFeedStop()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates *mediaDecodeSession* has an invalid value.

description `mpeos_mediaDripFeedStop()` should stop a drip-feed decode session.

related function(s) `mpeos_mediaDripFeedRenderFrame`
`mpeos_mediaDripFeedStart`

mpeos_mediaFreeze

freezes the media while presenting a single media frame.

syntax `mpe_Error mpeos_mediaFreeze(mpe_DispDevice videoDevice);`

parameter(s) `videoDevice` specifies the input video device to pause. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

value returned If the call is successful, `mpeos_mediaFreeze()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE EINVAL` indicates `videoDevice` has an invalid value.

`MPE_ERROR_MEDIA_OS` specifies an operating system error.

description `mpeos_mediaFreeze()` should capture and display a frame on the device while the media stream is still being decoded. A call to `mpeos_mediaResume()` should unfreeze the frame and resume displaying the media stream in the real-time position.

related function(s) `mpeos_mediaDecode`
`mpeos_mediaResume`
`mpeos_mediaStop`

mpeos_mediaFrequencyToTuner

identifies the tuner using the specified frequency.

syntax mpe_Error mpeos_mediaFrequencyToTuner(
 uint32_t *frequency*,
 uint32_t * *tunerId*);

parameter(s) *frequency* specifies the frequency to search.
tunerId is an output pointer to identify the tuner.

value returned If the call is successful, mpeos_mediaFrequencyToTuner() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *frequency* has an invalid value.

description mpeos_mediaFrequencyToTuner() should search for the specified frequency in hertz and return, through a pointer, the tuner identifier currently tuned to the specified frequency. If more than one tuner is set to the frequency, the first one found should be selected.

related function(s) mpeos_mediaGetTunerInfo

mpeos_mediaGetAFD

gets the current AFD for the decoder.

syntax `mpe_Error mpeos_mediaGetAFD(mpe_DispDevice decoder, mpe_MediaActiveFormatDescription * AFD);`

| | | |
|---------------------|-----------------------------------|---|
| parameter(s) | decoder | specifies the decoder. <code>mpe_DispDevice</code> is described in the <code>mpe_DispDevice</code> section. |
| | AFD | is an output pointer to the active format description variable. <code>mpe_MediaActiveFormatDescription</code> is described in the <code>mpe_MediaActiveFormatDescription</code> section. Currently, the following values are supported for <i>AFD</i> : |
| | <code>MPE_AFD_NOT_PRESENT</code> | indicates the format constant is not present. |
| | <code>MPE_AFD_16_9_TOP</code> | indicates the format is 16:9 (top). |
| | <code>MPE_AFD_14_9_TOP</code> | indicates the format is 14:9 (top). |
| | <code>MPE_AFD_GT_16_9</code> | indicates the format is greater than 16:9 (center). |
| | <code>MPE_AFD_SAME</code> | indicates the format is the same as the coded frame. |
| | <code>MPE_AFD_4_3</code> | indicates the format is 4:3 (center). |
| | <code>MPE_AFD_16_9</code> | indicates the format is 16:9 (center). |
| | <code>MPE_AFD_14_9</code> | indicates the format is 14:9 (center). |
| | <code>MPE_AFD_4_3_SP_14_9</code> | indicates the format is 4:3 (with shoot and protect 14:9 center). |
| | <code>MPE_AFD_16_9_SP_14_9</code> | indicates the format is 16:9 (with shoot and protect 14:9 center). |
| | <code>MPE_AFD_16_9_SP_4_3</code> | indicates the format is 16:9 (with shoot and protect 4:3 center). |

value returned If the call is successful, `mpeos_mediaGetAFD()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_ERROR_MEDIA_OS`
specifies an error occurred in the operating system.

description `mpeos_mediaGetAFD()` should get the current active format description for the decoder.

related function(s) none

mpeos_mediaGetAspectRatio

gets the aspect ratio for the video.

syntax `mpe_Error mpeos_mediaGetAspectRatio(mpe_DispDevice decoder, mpe_MediaAspectRatio * aspectRatio);`

parameter(s) `decoder` specifies the decoder. `mpe_DispDevice` is described in the `mpe_DispDevice` section.

`aspectRatio` is an output pointer to the aspect ratio. `mpe_MediaAspectRatio` is described in the `mpe_MediaAspectRatio` section. Currently, the following values are supported for `aspectRatio`:

`MPE_ASPECT_RATIO_UNKNOWN`
indicates an unknown ratio.

`MPE_ASPECT_RATIO_4_3`
indicates an 4:3 ratio.

`MPE_ASPECT_RATIO_16_9`
indicates an 16:9 ratio.

`MPE_ASPECT_RATIO_2_21_1`
indicates an 2.21:1 ratio.

value returned If the call is successful, `mpeos_mediaGetAspectRatio()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_ERROR_MEDIA_OS`
specifies an error occurred in the operating system.

description `mpeos_mediaGetAspectRatio()` should get the current broadcast aspect ratio for the specified decoder.

related function(s) none

mpeos_mediaGetBounds

gets the display dimensions for the target display device.

syntax `mpe_Error mpeos_mediaGetBounds(mpe_DispDevice videoDevice, mpe_MediaRectangle * srcRect, mpe_MediaRectangle * destRect);`

parameter(s) `videoDevice` specifies the display device. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

`srcRect` is an output pointer to origin and dimensions of a source rectangle. `mpe_MediaRectangle` is described in the *mpe_MediaRectangle* section.

`destRect` is an output pointer to origin and dimensions of a destination rectangle. `mpe_MediaRectangle` is described in the *mpe_MediaRectangle* section.

value returned If the call is successful, `mpeos_mediaGetBounds()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates `videoDevice` has an invalid value.

`MPE_ERROR_MEDIA_OS`
specifies an operating system error.

description `mpeos_mediaGetBounds()` should return the source and destination bounds for the target display device.

related function(s) `mpeos_mediaCheckBounds`
`mpeos_mediaSetBounds`

mpeos_mediaGetInputVideoSize

gets the video size.

syntax mpe_Error mpeos_mediaGetInputVideoSize(
 mpe_DispDevice *videoDevice*,
 mpe_GfxDimensions * *dim*);

parameter(s) *videoDevice* specifies the device in which to get the video size.
mpe_DispDevice is described in the *mpe_DispDevice* section.

dim is an output pointer to height and width in pixels of the video
before any scaling has taken place. *mpe_GfxDimensions* is
described in the *mpe_GfxDimensions* section.

value returned If the call is successful, *mpeos_mediaGetInputVideoSize()* should return
MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates *videoDevice* has an invalid value.

MPE_ENOMEM indicates the function failed due to insufficient memory
resources.

MPE_ERROR_MEDIA_OS
specifies an operating system error.

description *mpeos_mediaGetInputVideoSize()* should get the video pixel size before
setting up the scaling or clipping.

related function(s) none

mpeos_mediaGetScaling

gets the scaling capabilities of the specified decoder.

syntax `mpe_Error mpeos_mediaGetScaling(mpe_DispDevice decoder, mpe_MediaPositioningCapabilities * positioning, float ** horiz, float ** vert, mpe_Bool * hRange, mpe_Bool * vRange, mpe_Bool * canClip, mpe_Bool * supportsComponent);`

| | | |
|---------------------|--|---|
| parameter(s) | <i>decoder</i> | specifies the target decoder. <i>mpe_DispDevice</i> is described in the <i>mpe_DispDevice</i> section. |
| | <i>positioning</i> | is an output pointer to the returned value that specifies how the video can be positioned on the screen. <i>mpe_MediaPositioningCapabilities</i> is described in the <i>mpe_MediaPositioningCapabilities</i> section. Currently, the following values are supported for <i>positioning</i> : |
| | <code>MPE_POS_CAP_OTHER</code> | specifies the video positioning capability cannot be expressed by another <code>POS_CAP_*</code> constant. |
| | <code>MPE_POS_CAP_FULL</code> | specifies the video can be positioned anywhere on the screen, even if a part of the video is off screen. |
| | <code>MPE_POS_CAP_FULL_IF_ENTIRE_VIDEO_ON_SCREEN</code> | specifies the video can be positioned anywhere on screen as long as all the video is on screen. |
| | <code>MPE_POS_CAP_FULL_EVEN_LINES</code> | specifies the video can be positioned anywhere on the screen, even if a part of the video is off screen, with the restriction that the field order is respected. This implies that interlaced video can be positioned on even lines only (when counting from 0). |
| | <code>MPE_POS_CAP_FULL_EVEN_LINES_IF_ENTIRE_VIDEO_ON_SCREEN</code> | specifies the video can be positioned anywhere on screen as long as all the video is on screen, with the restriction that the field order is respected. |

| | |
|--------------------------|---|
| <i>horizontal</i> | is an output pointer to the returned list representing the discrete list of horizontal scaling factors available. The list can take one of two forms: |
| individual list | comprised of the available individual discrete scaling factors. In this case, the list should be terminated by a value of -1.0 (for example, a list of [1.0, 0.5, -1.0] to specify the decoder can scale at 100% or 50%). |
| two value list | comprised of an arbitrary range of scaling that is available (for example, a list of [0.5, 1.0] to specify the decoder can scale to any degree between 50% to 100%). |
| <i>vertical</i> | is an output pointer to the returned list representing the discrete list of vertical scaling factors available. The list can take one of two forms: |
| individual list | comprised of the available individual discrete scaling factors. In this case, the list should be terminated by a value of -1.0 (for example, a list of [1.0, 0.5, -1.0] to specify the decoder can scale at 100% or 50%). |
| two value list | comprised of an arbitrary range of scaling that is available (for example, a list of [0.5, 1.0] to specify the decoder can scale to any degree between 50% to 100%). |
| <i>hRange</i> | is an output pointer to the returned indicator to specify the form of the <i>horizontal</i> argument. If <i>hRange</i> is FALSE, <i>horizontal</i> is a list of discrete scaling factors (form I). If <i>hRange</i> is TRUE, <i>horizontal</i> is a range of arbitrary scaling factors. <i>mpe_Bool</i> is described in the <i>mpe_Bool</i> section. |
| <i>vRange</i> | is an output pointer to the returned indicator to specify the form of the <i>vertical</i> argument. If <i>vRange</i> is FALSE, <i>vertical</i> is a list of discrete scaling factors (form I). If <i>vRange</i> is TRUE, <i>vertical</i> is a range of arbitrary scaling factors. <i>mpe_Bool</i> is described in the <i>mpe_Bool</i> section. |
| <i>canClip</i> | is an output pointer to the returned value that indicates if the decoder supports clipping. If <i>canClip</i> is TRUE, the decoder can support clipping. If <i>canClip</i> is FALSE, the decoder cannot support clipping. <i>mpe_Bool</i> is described in the <i>mpe_Bool</i> section. |
| <i>supportsComponent</i> | is an output pointer to the returned value that indicates if the decoder supports component based scaling. If <i>supportsComponent</i> is TRUE, the decoder can support component based scaling. If <i>supportsComponent</i> is FALSE, the decoder cannot support component based scaling. <i>mpe_Bool</i> is described in the <i>mpe_Bool</i> section. |

value returned If the call is successful, `mpeos_mediaGetScaling()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:
`MPE_EINVAL` indicates *decoder* has an invalid value.

description `mpeos_mediaGetScaling()` should acquire the scaling capabilities of the specified decoder.

related function(s) none

mpeos_mediaGetTunerInfo

gets the current tuner information.

syntax mpe_Error mpeos_mediaGetTunerInfo(
 uint32_t tunerId,
 mpe_MediaTuneParams * tuneParams);

| | | |
|---------------------|-------------------|---|
| parameter(s) | <i>tunerId</i> | specifies the tuner. |
| | <i>tuneParams</i> | is an output pointer to the tuner information. <i>mpe_MediaTuneParams</i> is described in the <i>mpe_MediaTuneParams</i> section. |

value returned If the call is successful, `mpeos_mediaGetTunerInfo()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates *tunerId* has an invalid value.

`MPE_ERROR_MEDIA_OS`
specifies an operating system error.

description `mpeos_mediaGetTunerInfo()` should return the current tuning parameters for the target tuner. If the tuner is idle (neither tuning nor tuned) the frequency parameter should be set to 0. If tuned to an analog channel, the qamMode should be set to `MPE_MEDIA_QAM_MODE_NTSC`.

`mpeos_mediaGetTunerInfo()` is currently called from the Object Carousel manager in the OCAP stack, and the Section Filter manager in the MPEOS layer to verify that tuning parameters passed to them in setup requests are consistent with the actual current tuner settings.

related function(s) `mpeos_mediaFrequencyToTuner`

mpeos_mediaInit

initializes the media API.

syntax mpe_Error mpeos_mediaInit(void);

parameter(s) none

value returned If the call is successful, mpeos_mediaInit() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description mpeos_mediaInit() should perform platform-specific initialization support for the media API.

related function(s) mpeos_mediaShutdown

mpeos_mediaRegisterQueueForTuneEvents

registers an event queue.

syntax mpe_Error mpeos_mediaRegisterQueueForTuneEvents(
 mpe_EventQueue *queueId*);

parameter(s) *queueId* identifies an additional queue that receives decode and tune events that result from any mpeos_mediaDecode() or mpeos_mediaTune() call. mpe_EventQueue is described in the *mpe_EventQueue* section. Currently, the following events are supported:

- ◆ **NOTE:** If an error is returned, no future event are delivered.

MPE_TUNE_ABORT

specifies either a tuning parameter change or an asynchronous failure occurred during or after tuning.

MPE_TUNE_COMPLETE

specifies the tune is complete.

MPE_TUNE_FAIL

specifies the tune did not complete because the tuner was unable to tune for any reason.

MPE_TUNE_STARTED

specifies a tune was requested and that request is being processed.

value returned If the call is successful, mpeos_mediaRegisterQueueForTuneEvents() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates *queueId* has an invalid value.

MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description mpeos_mediaRegisterQueueForTuneEvents() should register an MPE event queue for receiving all tuner events.

related function(s) mpeos_mediaDecode
 mpeos_mediaTune
 mpeos_mediaUnregisterQueue

mpeos_mediaResume

resumes the media decoding.

syntax mpe_Error mpeos_mediaResume(mpe_DispDevice videoDevice);

parameter(s) *videoDevice* specifies the input video device to resume. *mpe_DispDevice* is described in the *mpe_DispDevice* section.

value returned If the call is successful, *mpeos_mediaResume()* should return *MPE_SUCCESS*. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates *videoDevice* has an invalid value.

MPE_ERROR_MEDIA_OS specifies an operating system error.

description *mpeos_mediaResume()* should remove the frozen frame from the screen and resume the media stream.

related function(s) *mpeos_mediaDecode*
mpeos_mediaFreeze
mpeos_mediaStop

mpeos_mediaSetBounds

sets the bounds of a display device.

syntax `mpe_Error mpeos_mediaSetBounds(`
`mpe_DispDevice videoDevice,`
`mpe_MediaRectangle * srcRect,`
`mpe_MediaRectangle * destRect);`

parameter(s) `videoDevice` specifies the display device. `mpe_DispDevice` is described in the *mpe_DispDevice* section.

`srcRect` is an input pointer to origin and dimensions of a source rectangle. `mpe_MediaRectangle` is described in the *mpe_MediaRectangle* section.

`destRect` is an input pointer to origin and dimensions of a destination rectangle. `mpe_MediaRectangle` is described in the *mpe_MediaRectangle* section.

value returned If the call is successful, `mpeos_mediaSetBounds()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE_ERROR_MEDIA_OS`
 specifies an operating system error.

description `mpeos_mediaSetBounds()` should set the source and destination bounds for the target display device.

related function(s) `mpeos_mediaCheckBounds`
`mpeos_mediaGetBounds`

mpeos_mediaShutdown

sends shut down events to the tuner and decoder.

syntax mpe_error mpeos_mediaShutdown(void);

parameter(s) none

value returned If the call is successful, mpeos_mediaShutdown() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_ERROR_MEDIA_OS
specifies an operating system error.

description mpeos_mediaShutdown() should send shutdown events to the tuner and decoder. mpeos_mediaShutdown() releases any media-related resources back to the operating system. mpeos_mediaShutdown() sends the MPE_EVENT_SHUTDOWN event to the queue identifier of mpeos_mediaTune() and mpeos_mediaDecode().

related function(s) mpeos_mediaInit

mpeos_mediaStop

stops presenting the media.

syntax `mpe_Error mpeos_mediaStop(mpe_MediaDecodeSession session);`

parameter(s) `session` specifies the video-drip feed session.

value returned If the call is successful, `mpeos_mediaStop()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates `session` has an invalid value.

`MPE_ERROR_MEDIA_RESOURCE_NOT_ACTIVE`
specifies the resource is not available.

`MPE_ERROR_MEDIA_OS`
specifies an operating system error.

description `mpeos_mediaStop()` should stop decoding and presenting the media.
`mpeos_mediaStop()` sends the `MPE_EVENT_SHUTDOWN` event to the queue identifier of `mpeos_mediaDecode()`.

related function(s) `mpeos_mediaDecode`
`mpeos_mediaFreeze`
`mpeos_mediaResume`

mpeos_mediaSwapDecoders

swaps the video source feeds to the two specified decoders.

syntax `syntax mpe_Error mpeos_mediaSwapDecoders(
 mpe_DispDevice decoder1,
 mpe_DispDevice decoder2,
 mpe_Bool useAudio);`

| | | |
|---------------------|-----------------|---|
| parameter(s) | <i>decoder1</i> | specifies one of the two specified decoders to swap. <i>mpe_DispDevice</i> is described in the <i>mpe_DispDevice</i> section. |
| | <i>decoder2</i> | specifies one of the two specified decoders to swap. <i>mpe_DispDevice</i> is described in the <i>mpe_DispDevice</i> section. |
| | <i>useAudio</i> | specifies which audio to present after the swap. If <i>useAudio</i> is TRUE, the audio associated with the <i>decoder1</i> stream is presented. If <i>useAudio</i> is FALSE, the audio associated with the <i>decoder2</i> is presented. <i>mpe_Bool</i> is described in the <i>mpe_Bool</i> section. |

| | |
|-----------------------|--|
| value returned | If the call is successful, <code>mpeos_mediaSwapDecoders()</code> should return MPE_SUCCESS. Otherwise, it should return the following error code: |
| MPE_EINVAL | indicates at least one input parameter to the function has an invalid value. |

description `mpeos_mediaSwapDecoders()` should be used to swap the video source stream feeding two specified decoders. An example of its use would be (but is not limited to) the situation with swapping the current window and the picture-in-picture window.

related function(s) `mpeos_mediaDecode`

mpeos_mediaTune

starts the tune operation.

```
syntax mpe_Error mpeos_mediaTune (
    mpe_MediaTuneRequestParams * tuneRequest,
    mpe_EventQueue queueId ,
    void * act );
```

parameter(s) *tuneRequest* is an input pointer to the tune request parameters. *mpe_MediaTuneRequestParams* are defined in the *mpe_MediaTuneRequestParams* section.

queueId identifies the queue in which to send the tune events associated with this tune request. *mpe_EventQueue* is described in the *mpe_EventQueue* section. Currently, the following events are supported:

- ◆ **NOTE:** If an error is returned, no future event are delivered.

MPE_EVENT_SHUTDOWN

indicates no further events can be sent to the queue.

MPE_TUNE_ABORT

specifies either a tuning parameter change or an asynchronous failure occurred during or after tuning.

MPE_TUNE_COMPLETE

specifies the tune is complete.

MPE_TUNE_FAIL

specifies the tune did not complete because the tuner was unable to tune for any reason.

MPE_TUNE_STARTED

specifies a tune has been requested and that request is being processed.

act

is an input pointer to an ACT. When the asynchronous events are sent to the event queue specified in *queueId* (via *mpeos_eventQueueSend()*), the event should pass this *act* pointer in the *optionalEventData2* parameter. The pointer to *act* should not be sent to those event queues which are registered via *mpeos_mediaRegisterQueueForTuneEvents()*.

value returned If the call is successful, `mpeos_mediaTune()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE_ERROR_MEDIA_BAD_TUNING_REQUEST` specifies the tune request failed.

`MPE_ERROR_MEDIA_INVALID_ID` specifies an invalid tuner identifier.

`MPE_ERROR_MEDIA_OS` specifies an operating system error.

description `mpeos_mediaTune()` should start the asynchronous tune operation. An error code is synchronously returned to the caller to indicate a preliminary state of resource allocation.

When tuning to an analog service, specifying the frequency alone is sufficient (QAM mode = 0, `program_number` is not applicable). For more information about QAM modes, refer to the *mpe_SiModulationMode* section.

related function(s) `mpeos_eventQueueNew`
`mpeos_mediaDecode`

mpeos_mediaUnregisterQueue

unregisters an event queue.

syntax mpe_Error mpeos_mediaUnregisterQueue(mpe_EventQueue queueId);

parameter(s) *queueId* identifies the queue to unregister. *mpe_EventQueue* is described in the *mpe_EventQueue* section.

value returned If the call is successful, *mpeos_mediaUnregisterQueue()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *queueId* has an invalid value.

description *mpeos_mediaUnregisterQueue()* should unregister an MPE event queue from receiving tuning and decoding events. Tuning and decoding events are defined in the *Events* section.

related function(s) *mpeos_mediaDecode*
mpeos_mediaRegisterQueueForTuneEvents
mpeos_mediaTune

MEM Memory API

Overview

The memory Application Programming Interface (API) provides the basic support for allocating and deallocating memory, as well as some additional support for basic system memory status information.

The memory API provides the ability to allocate and deallocate blocks of memory by pointer or by handle.

The `mpeos_memCompact`, `mpeos_memLockHandle` and `mpeos_memPurge` functions manipulate memory allocated by handle.

The `mpeos_memGetFreeSize`, `mpeos_memGetLargestFree`, `mpeos_memGetStats`, and `mpeos_memStats` functions allow for acquiring basic information on the current status of system memory, assuming the operating system provides support for these actions.

Before reading this chapter, you should be familiar with:

- ◆ the semantics of the module support API
- ◆ the use of the MPE initialization module to provide
- ◆ the memory configuration parameters
- ◆ the concepts of handle versus pointer based memory allocation schemes

After reading this chapter, you should be able to:

- ◆ port the memory functionality within the OCAP stack
- ◆ configure the MPE initialization module memory parameters

Memory allocation failure handling

If a memory allocation failure occurs within the OCAP stack, it is desirable to try and recover, and in the worst case, fail as gracefully as possible. When an allocation via `mpeos_memAllocP` fails, the implementation is expected to attempt to recover any memory that is deemed less than critical. Certain areas of the stack allocate memory which is not critical to the execution of the stack (generally, for performance enhancements), and if needed, can free up this memory in order to allow other allocations to continue. To this end, this API defines the `mpeos_memRegisterMemFreeCallback()` function.

With `mpeos_memRegisterMemFreeCallback()`, the stack registers certain callback routines that can be called in case a memory allocation initially fails. These callback routines are passed information about the allocation that failed (namely, the color and amount of memory that was requested), along with an extra piece of information supplied at the time the callback function was registered. Callback routines return one of the following values:

- 0 indicates the callback routine was unsuccessful at returning memory to the requested memory heap.
- 1 indicates the callback routine was successful at returning an unknown amount of memory to the requested memory heap.

num (between 1 and *maximum int*) 1

indicates the callback was successful at returning a number of bytes of memory to the requested memory heap.

`mpeos_memAllocPGen()` is expected to execute the following algorithm:

```

ret_val = color-appropriate memory allocation (requested allocation size)
)

if (fail) {
    for each callback routine registered for the requested color, or
    equivalent color {
        x = call callback routine (requested color, requested amount,
abstract data)
        if (x <> 0)
            ret_val = color-appropriate memory allocation (requested
allocation size)
            if (success)
                return
    }
}
```

```
for each callback routine registered for  
MPE_MEM_CALLBACK_SYSEVENT {  
    x = call callback routine ( requested color, requested amount,  
      abstract data )  
    if (x <> 0)  
        ret_val = color appropriate memory allocation ( requested  
          allocation size )  
        if (success)  
            return  
    }  
  
for each callback routine registered for  
MPE_MEM_CALLBACK_APPKILL {  
    x = call callback routine ( requested color, requested amount,  
      abstract data )  
    if (x <> 0)  
        ret_val = color-appropriate memory allocation ( requested  
          allocation size )  
        if (success)  
            return  
        else  
            get next x  
    }
```

If all of these routines fail to free enough of the requested memory, `mpeos_memAllocPGen()` sets the return value to `NULL` and returns the appropriate error message.

Definitions

The following definitions, which are defined in `mpeos_mem.h`, are used by the memory functions:

| | |
|--------------------------------|---------------------------------|
| <code>MPE_MEM_NOPURGE</code> | <code>MPE_MEM_PRIOR_HIGH</code> |
| <code>MPE_MEM_PRIOR_LOW</code> | <code>mpeos_memAllocP</code> |
| <code>mpeos_memFreeP</code> | <code>mpeos_memReallocP</code> |

`MPE_MEM_NOPURGE`

`MPE_MEM_NOPURGE` indicates an unpurgeable priority. `MPE_MEM_NOPURGE` is defined as follows:

```
#define MPE_MEM_NOPURGE 0
```

`MPE_MEM_PRIOR_HIGH`

`MPE_MEM_PRIOR_HIGH` indicates the highest purgeable priority. `MPE_MEM_PRIOR_HIGH` is defined as follows:

```
#define MPE_MEM_PRIOR_HIGH 1
```

The highest priority is represented by the smallest value, and the lowest priority is represented by the largest value. In other words, the lower the value, the higher the priority.

`MPE_MEM_PRIOR_LOW`

`MPE_MEM_PRIOR_LOW` indicates the lowest purgeable priority. `MPE_MEM_PRIOR_LOW` is defined as follows:

```
#define MPE_MEM_PRIOR_LOW 255
```

`mpeos_memAllocP`

`mpeos_memAllocP` is the generic memory allocation routine. `mpeos_memAllocP` is defined as a macro that maps to either `mpeos_memAllocPGen` or `mpeos_memAllocPProf`. When profiling is enabled, as specified by the definition of `MPE_FEATURE_MEM_PROF`, `mpeos_memAllocP` maps to `mpeos_memAllocPProf`. When profiling is disabled, `mpeos_memAllocP` maps to `mpeos_memAllocPGen`. `mpeos_memAllocP` is defined as follows:

```
#if defined (MPE_FEATURE_MEM_PROF)
# define mpeos_memAllocP(c,s,m) mpeos_memAllocPProf
    (c, s, m, __FILE__, __LINE__)
#else
# define mpeos_memAllocP mpeos_memAllocPGen
#endif
```

mpeos_memFreeP

`mpeos_memFreeP` is the generic memory de-allocation routine.
`mpeos_memFreeP` is defined as a macro that maps to either
`mpeos_memFreePGen` or `mpeos_memFreePProf`. When profiling is enabled, as
specified by the definition of `MPE_FEATURE_MEM_PROF`, `mpeos_memFreeP` maps
to `mpeos_memFreePProf`. When profiling is disabled, `mpeos_memFreeP` maps
to `mpeos_memFreePGen`. `mpeos_memFreeP` is defined as follows:

```
#ifdef MPE_FEATURE_MEM_PROF
#define mpeos_memFreeP(c,m) mpeos_memFreePProf
    (c, m, __FILE__, __LINE__)
#else
#define mpeos_memFreeP mpeos_memFreePGen
#endif
```

mpeos_memReallocP

`mpeos_memReallocP` is the generic memory reallocation routine.
`mpeos_memReallocP` is defined as a macro that maps to either
`mpeos_memReallocPGen` or `mpeos_memReallocPProf`. When profiling is enabled, as
specified by the definition of `MPE_FEATURE_MEM_PROF`, `mpeos_memReallocP` maps to
`mpeos_memReallocPProf`. When profiling is disabled, `mpeos_memReallocP` maps to
`mpeos_memReallocPGen`. `mpeos_memReallocP` is defined as follows:

```
#ifdef MPE_FEATURE_MEM_PROF
#define mpeos_memReallocP(c,s,m) mpeos_memReallocPProf
    (c, s, m, __FILE__, __LINE__)
#else
#define mpeos_memReallocP mpeos_memReallocPGen
#endif
```

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the memory functions:

MPE_EINVAL
MPE_ENOMEM

MPE_ENODATA
MPE_SUCCESS

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

MPE_EINVAL

MPE_EINVAL indicates at least one input parameter to the function has an invalid value. MPE_EINVAL is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENODATA

MPE_ENODATA indicates no data is available for the specified request.

MPE_ENODATA is defined as follows:

```
#define MPE_ENODATA OS_ENODATA
```

MPE_ENOMEM

MPE_ENOMEM indicates the function failed due to insufficient memory resource. MPE_ENOMEM is defined as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_SUCCESS

MPE_SUCCESS indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). MPE_SUCCESS is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpeos_mem.h`, `mpe_types.h`, and `mpe_error.h`, are used by the memory functions:

| | |
|-------------------------------|----------------------------|
| <code>mpe_Bool</code> | <code>mpe_Error</code> |
| <code>mpe_MemColor</code> | <code>mpe_MemHandle</code> |
| <code>mpe_MemStatsInfo</code> | |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t_t mpe_Error;
```

`mpe_MemColor`

`mpe_MemColor` specifies the intended use of allocated memory. Color is a loose designation of the type or intended use of the memory. Color may or may not be used depending on implementation. For example, Color may map to a distinct heap. For example, the uses of the implementation are:

- ◆ to trace memory use
- ◆ to indicate distinct allocation heaps
- ◆ to enable purging/compaction of one type of memory

Adding new memory allocations may be necessary. Generally, at least one memory allocation is added for every logical code module and more can be added for additional sub-areas per code module. It is preferred that this total is kept under 32 to allow use in a bit-field. `mpe_MemColor` is defined in `mpe_types.h` as follows:

```
typedef enum {
    MPE_MEM_SYSTEM = -1,
    MPE_MEM_GENERAL = 0,
    MPE_MEM_TEMP,
    MPE_MEM_JVM,
    MPE_MEM_GFX,
    MPE_MEM_GFX_LL,
    MPE_MEM_SYNC,
    MPE_MEM_THREAD,
    MPE_MEM_FILE,
    MPE_MEM_FILE_CAROUSEL,
    MPE_MEM_MEDIA,
    MPE_MEM_UTIL,
    MPE_MEM_EVENT,
    MPE_MEM_FILTER,
    MPE_MEM_POD,
    MPE_MEM_SI,
    MPE_MEM_TEST,
    MPE_MEM_NET,
```

```

        MPE_MEM_CC,
        MPE_MEM_DVR,
        MPE_MEM SND,
        MPE_MEM FP,
        MPE_MEM_PORT,
        MPE_MEM_STORAGE,
        MPE_MEM_VBI,
        PORT_MEM_COLORS_COMMA
        MPE_MEM_NCOLORS
    } mpe_MemColor;

```

where

MPE_MEM_SYSTEM

specifies system-wide memory. Use with `GetFreeSize()` and `GetLargestFree()` only.

MPE_MEM_GENERAL

specifies non-specific memory.

MPE_MEM_TEMP specifies temporary memory.

MPE_MEM_JVM specifies memory used for the Java Virtual Machine (JVM), including system and Java heaps.

MPE_MEM_GFX specifies memory used for generic graphics.

MPE_MEM_GFX_LL

specifies memory used for low-level graphics. For example, DirectFB, which is the intermediate, portable graphics package used in MPE.

MPE_MEM_SYNC specifies memory used for synchronization primitives.

MPE_MEM_THREAD

specifies memory used for threads.

MPE_MEM_FILE specifies memory used for the generic file system.

MPE_MEM_FILE_CAROUSEL

specifies memory used for data and object carousel.

MPE_MEM_MEDIA

specifies memory used for generic media.

MPE_MEM_UTIL specifies memory used for utilities.

MPE_MEM_EVENT

specifies memory used for events.

MPE_MEM_FILTER

specifies memory used for section filtering.

MPE_MEM_POD specifies memory used for CableCard (Point-Of-Deployment (POD)).

MPE_MEM_SI specifies memory used for service information.

MPE_MEM_TEST specifies memory used for testing.

MPE_MEM_NET specifies memory used for networking.

MPE_MEM_CC specifies memory used for closed captioning.

MPE_MEM_DVR specifies memory used for Digital Video Recorder (DVR).

MPE_MEM SND specifies memory used for sound.

MPE_MEM FP specifies memory used for the front panel.

MPE_MEM_PORT specifies memory used for port-specific support.

MPE_MEM_STORAGE
specifies memory used for storage.

MPE_MEM_VBI specifies memory used for Vertical Blanking Interval (VBI) filtering.

POR T_MEM_COLORS_COMM A
specifies a macro for port-specific color definitions using
PORT_MEM_COLORS.PORT_MEM_COLORS_COMM A is not an actual color.
PORT_MEM_COLORS is defined as follows:

PORT_MEM_COLORS
is define by a port (via os_types.h) to provide
port-specific memory color definitions. If defined,
the additional color symbols become part of
mpe_MemColor, being inserted after all pre-defined
colors and before MPE_MEM_NCOLORS. Following is an
example definition of PORT_MEM_COLORS:

```
#define PORT_MEM_COLORS PORT_MEM_BLUE,
    PORT_MEM_RED, PORT_MEM_PURPLE
```

MPE_MEM_NCOLORS
specifies the number of colors defined.

mpe_MemHandle

mpe_MemHandle specifies an opaque type used to represent a handle to allocated memory. Handles should be used to indicate the allocated memory can be purged and/or relocated. mpe_MemHandle is defined in mpe_types.h as follows:

```
typedef struct { int unused; } * mpe_MemHandle;
```

mpe_MemStatsInfo

mpe_MemStatsInfo specifies the information related to a specific memory color. The structure returns in an array from mpeos_memGetStats(). mpe_MemStatsInfo is defined in mpeos_mem.h as follows:

```
typedef struct mpe_MemStatsInfo {
    const char *name;
    uint32_t currAllocated;
    uint32_t maxAllocated;
} mpe_MemStatsInfo;
```

where

| | |
|------|--|
| name | specifies the ASCII, NULL-terminated string for the name of the memory color. The name for the color value is usually the same as the mpe_MemColor, enum member without the MPE_MEM_ prefix. |
|------|--|

currAllocated

specifies the current number of bytes currently allocated to the memory color.

maxAllocated specifies the highest number of bytes that has ever been allocated for the memory color.

Supported functions

The memory functions allocate and free the memory. The following memory functions need to be supported:

- `mpeos_memAllocH`
allocates a block of memory via a handle.
- `mpeos_memAllocPGen`
allocates a block of memory via a pointer.
- `mpeos_memAllocPProf`
allocates a block of memory via a pointer for testing memory functions.
- `mpeos_memCompact`
attempts to compact memory.
- `mpeos_memFreeH`
frees a block of memory via a handle.
- `mpeos_memFreePGen`
frees a block of memory via a pointer.
- `mpeos_memFreePProf`
frees a block of memory via a pointer for testing memory functions.
- `mpeos_memGetFreeSize`
gets a specified size of free system memory.
- `mpeos_memGetLargestFree`
gets the largest block of free system memory.
- `mpeos_memGetStats`
copies memory color statistics into an array.
- `mpeos_memInit`
initializes operating system memory.
- `mpeos_memLockH`
locks or unlocks the data referenced by a handle.
- `mpeos_memPurge`
attempts to purge memory.
- `mpeos_memReallocH`
resizes the block of memory via a handle.
- `mpeos_memReallocPGen`
resizes the block of memory via a pointer.
- `mpeos_memReallocPProf`
resizes the memory block for testing memory functions.

`mpeos_memRegisterMemFreeCallback`
registers a function to call in case of memory exhaustion.

`mpeos_memStats`
outputs the current internal memory statistics.

`mpeos_memUnregisterMemFreeCallback`
unregisters a callback to be called when memory is low.

mpeos_memAllocH

allocates a block of memory via a handle.

syntax

```
mpe_Error mpeos_memAllocH(
    mpe_MemColor color,
    uint32_t size,
    uint32_t priority,
    mpe_MemHandle * handle );
```

parameter(s)

| | |
|--------------------|---|
| <i>color</i> | specifies the type or intended use of the memory, depending on the implementation. <i>mpe_MemColor</i> is defined in the <i>mpe_MemColor</i> section. |
| <i>size</i> | specifies the size, in bytes, of the memory block to allocate. |
| <i>priority</i> | indicates the priority. Purgeable handles with lower priority are more likely to be purged than higher. Currently, the following values are supported for <i>priority</i> : |
| MPE_MEM_NOPURGE | indicates an unpurgeable priority. |
| MPE_MEM_PRIOR_HIGH | indicates the highest purgeable priority. |
| MPE_MEM_PRIOR_LOW | indicates the lowest purgeable priority. |
| <i>handle</i> | is an output pointer for returning the handle to the newly allocated memory block. <i>mpe_MemHandle</i> is defined in the <i>mpe_MemHandle</i> section. |

value returned

If the call is successful, *mpeos_memAllocH()* should return *MPE_SUCCESS*, along with a pointer to the new memory block's handle. Otherwise, it should return one of the following error codes:

| | |
|------------|--|
| MPE EINVAL | indicates at least one input parameter to the function has an invalid value. |
| MPE ENOMEM | indicates the function failed due to insufficient memory resources. |

description

mpeos_memAllocH() should allocate a block of system memory that is returned by a *handle*. The handle-referenced memory may be purged or relocated if supported by the implementation. The size of the memory is specified by *size*. The address of the allocated memory block's handle is specified by the *handle* pointer.

related function(s)

- mpeos_memCompact*
- mpeos_memFreeH*
- mpeos_memLockH*
- mpeos_memPurge*
- mpeos_memReallocH*

mpeos_memAllocPGen

allocates a block of memory via a pointer.

syntax `mpe_Error mpeos_memAllocPGen(`
`mpe_MemColor color,`
`uint32_t size,`
`void ** memory);`

| | | |
|---------------------|---------------|---|
| parameter(s) | <i>color</i> | specifies the type or intended use of the memory, depending on the implementation. <code>mpe_MemColor</code> is defined in the <i>mpe_MemColor</i> section. |
| | <i>size</i> | specifies the size, in bytes, of the memory block to allocate. |
| | <i>memory</i> | is an output pointer for returning the pointer to the newly-allocated memory block. |

value returned If the call is successful, `mpeos_memAllocPGen()` should return `MPE_SUCCESS`, along with a pointer to the new memory block specified by *memory*. Otherwise, it should return one of the following error codes:

| | |
|-------------------------|--|
| <code>MPE EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| <code>MPE ENOMEM</code> | indicates the function failed due to insufficient memory resources. |

description `mpeos_memAllocPGen()` should allocate a block of system memory specified by *size*. The address of the allocated memory block is returned by the *memory* pointer. `mpeos_memAllocPGen()` is used for general purpose (normal) functions.

The `mpeos_memAllocP` macro, as specified by the definition of `MPE_FEATURE_MEM_PR0F`, maps to `mpeos_memAllocPGen()` when profiling is disabled. `mpeos_memAllocP` is described in the *mpeos_memAllocP* section.

related function(s) `mpeos_memCompact`
`mpeos_memFreePGen`
`mpeos_memPurge`
`mpeos_memReallocPGen`

mpeos_memAllocPProf

allocates a block of memory via a pointer for testing memory functions.

syntax `mpe_Error mpeos_memAllocPProf(mpe_MemColor color, uint32_t size, void **memory, char *fileName, uint32_t lineNumber);`

| | | |
|---------------------|-----------------|---|
| parameter(s) | <i>color</i> | specifies the type or intended use of the memory, depending on the implementation. <code>mpe_MemColor</code> is defined in the <i>mpe_MemColor</i> section. |
| | <i>size</i> | specifies the size, in bytes, of the memory block to allocate. |
| | <i>memory</i> | is an output pointer for returning the pointer to the newly allocated memory block. |
| | <i>fileName</i> | is a pointer to the file name. |
| | <i>lineNum</i> | specifies the line number in the file. |

| | |
|-----------------------|--|
| value returned | If the call is successful, <code>mpeos_memAllocPProf()</code> should return <code>MPE_SUCCESS</code> , along with a pointer to the new memory block specified by <i>memory</i> . Otherwise, it should return one of the following error codes: |
| | <code>MPE EINVAL</code> indicates at least one input parameter to the function has an invalid value. |
| | <code>MPE ENOMEM</code> indicates the function failed due to insufficient memory resources. |

| | |
|--------------------|--|
| description | <code>mpeos_memAllocPProf()</code> should allocate a block of system memory of the specified <i>size</i> . The address of the memory block allocated is returned via the <i>memory</i> pointer. <code>mpeos_memAllocPProf()</code> is used for profiling (testing) versions of the memory functions. The <code>mpeos_memAllocP</code> macro, as specified by the definition of <code>MPE_FEATURE_MEM_PROF</code> , maps to <code>mpeos_memAllocPProf()</code> when profiling is enabled. <code>mpeos_memAllocP</code> is described in the <i>mpeos_memAllocP</i> section. |
|--------------------|--|

- ◆ **NOTE:** `mpeos_memAllocPProf()` should not be used in production code, because there is run-time and memory overhead associated with the memory profiling.

| | |
|----------------------------|---|
| related function(s) | <code>mpeos_memFreePProf</code> <code>mpeos_memReallocPProf</code> |
|----------------------------|---|

mpeos_memCompact

attempts to compact memory.

syntax mpe_Error mpeos_memCompact(mpe_MemColor color);

parameter(s) *color* specifies the type or intended use of the memory, depending on the implementation. *mpe_MemColor* is defined in the *mpe_MemColor* section.

value returned If the call is successful, *mpeos_memCompact()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_memCompact()* should attempt to compact a block of system memory. If the implementation supports it, non-locked and relocatable handle-based allocations are rearranged to provide the largest amount of free-space, in bytes, available. If the implementation does not support purging, *mpeos_memCompact()* does nothing except return MPE_SUCCESS.

related function(s) *mpeos_memReallocH*

mpeos_memFreeH

frees a block of memory via a handle.

syntax mpe_Error mpeos_memFreeH(
 mpe_MemColor color,
 mpe_MemHandle handle);

parameter(s) *color* specifies the type or intended use of the memory, depending on the implementation. *mpe_MemColor* is defined in the *mpe_MemColor* section.

handle is a handle, returned by *mpeos_memAllocH()* or *mpeos_memReallocH()*, for the memory block to free. *mpe_MemHandle* is defined in the *mpe_MemHandle* section.

value returned If the call is successful, *mpeos_memFreeH()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_memFreeH()* should free the block of system memory specified by *handle*.

related function(s) *mpeos_memAllocH*
mpeos_memCompact
mpeos_memLockH
mpeos_memPurge
mpeos_memReallocH

mpeos_memFreePGen

frees a block of memory via a pointer.

syntax mpe_Error mpeos_memFreePGen(
 mpe_MemColor color,
 void * memory);

parameter(s) *color* specifies the original color specified on allocation.
mpe_MemColor is defined in the *mpe_MemColor* section.
memory is an input pointer to the memory block to free.

value returned If the call is successful, *mpeos_memFreePGen()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:
MPE EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_memFreePGen()* should free the block of system memory specified by *memory*.
The *mpeos_memFreeP* macro, as specified by the definition of *MPE_FEATURE_MEM_PROF*, maps to *mpeos_memFreePGen()* when profiling is disabled. *mpeos_memFreeP* is described in the *mpeos_memFreeP* section.

related function(s) *mpeos_memAllocPGen*
mpeos_memReallocPGen

mpeos_memFreePProf

frees a block of memory via a pointer for testing memory functions.

syntax

```
mpe_Error mpeos_memFreePProf(
    mpe_MemColor color,
    void * memory,
    char * fileName,
    uint32_t lineNumber );
```

| | | |
|---------------------|-----------------|---|
| parameter(s) | <i>color</i> | specifies the type or intended use of the memory, depending on the implementation. <i>mpe_MemColor</i> is defined in the <i>mpe_MemColor</i> section. |
| | <i>memory</i> | a pointer to the memory block to free. |
| | <i>fileName</i> | is an input pointer to the file name. |
| | <i>lineNum</i> | specifies the line number in the file. |

value returned If the call is successful, *mpeos_memFreePProf()* should return *MPE_SUCCESS*, along with a pointer to the new memory block specified by *memory*. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_memFreePProf()* should free the specified block of system memory. *mpeos_memFreePProf()* is used for profiling (testing) versions of the memory functions.

The *mpeos_memFreeP* macro, as specified by the definition of *MPE_FEATURE_MEM_PROF*, maps to *mpeos_memFreePProf()* when profiling is enabled. *mpeos_memFreeP* is described in the *mpeos_memFreeP* section.

- ◆ **NOTE:** *mpeos_memFreePProf()* should not be used in production code, because there is run-time and memory overhead associated with the memory profiling.

related function(s) *mpeos_memAllocPProf*
mpeos_memReallocPProf

mpeos_memGetSize

gets a specified size of free system memory.

syntax mpe_Error mpeos_memGetSize(
 mpe_MemColor *color*,
 uint32_t * *freeSize*);

parameter(s) *color* specifies the type or intended use of the memory, depending on the implementation. *mpe_MemColor* is defined in the *mpe_MemColor* section.

freeSize is an output pointer for returning the size, in bytes, of the free memory in the system.

value returned If the call is successful, *mpeos_memGetSize()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *color* has an invalid value.

description *mpeos_memGetSize()* should get the amount of free system memory for the specified color and write it at the pointer, specified by *freeSize*.

related function(s) *mpeos_memGetLargestFree*

mpeos_memGetLargestFree

gets the largest block of free system memory.

syntax mpe_Error mpeos_memGetLargestFree(
 mpe_MemColor *color*,
 uint32_t * *freeSize*);

parameter(s) *color* specifies the type or intended use of the memory, depending on the implementation. *mpe_MemColor* is defined in the *mpe_MemColor* section.

freeSize is an output pointer for returning the size, in bytes, of the largest free memory block in the system.

value returned If the call is successful, *mpeos_memGetLargestFree()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE_EINVAL indicates *color* has an invalid value.

description *mpeos_memGetLargestFree()* should get the size of the largest block of free system memory.

related function(s) *mpeos_memGetFreeSize*

mpeos_memGetStats

copies memory color statistics into an array.

syntax mpe_Error mpeos_memGetStats(
 uint32_t size,
 mpe_MemStatsInfo * stats);

parameter(s) *size* specifies the total number of bytes pointed to by *stats*. *size* should be equal to `sizeof(mpe_MemStatsInfo) * MPE_MEM_NCOLORS`.

stats is an output pointer to an array used to store the memory statistics for every memory color. *mpe_MemStatsInfo* is defined in the *mpe_MemStatsInfo* section.

value returned If the call is successful, `mpeos_memGetStats()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates the *size* is not the size of *mpe_MemStatsInfo* times `MPE_MEM_NCOLORS` or the implementation does not support the tracking of memory statistics .

description `mpeos_memGetStats()` should get the memory statistics for all the colors in the array of *mpe_MemStatsInfo*.

related function(s) none

mpeos_memInit

initializes operating system memory.

syntax void mpeos_memInit(void);

parameter(s) none

value returned none

description mpeos_memInit() should perform any low-level setup required to initialize the operating system memory. mpeos_memInit() performs any low-level initialization required to set up the operating-system-level memory subsystem. The memory subsystem should be initialized following the environment subsystem (via mpeos_envInit()) to allow it to be configured by the environment. However, access to the memory APIs prior to initialization (for example, by mpeos_envInit()) should be allowed.

related function(s) none

mpeos_memLockH

locks or unlocks the data referenced by a handle.

syntax `mpe_Error mpeos_memLockH(
 mpe_MemHandle handle,
 void ** address);`

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>handle</i> | specifies the handle to a block of memory that should be considered locked. <code>mpe_MemHandle</code> is defined in the <i>mpe_MemHandle</i> section. |
| | <i>address</i> | is an output pointer to a pointer where the address of the relevant data block is written following a successful locking of the data block. If <i>address</i> is NULL, the previous locking of <i>handle</i> should be reversed. |

value returned If the call is successful, `mpeos_memLockH()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates *handle* has an invalid value.

description `mpeos_memLockH()` should lock or unlock the data referenced by the *handle*. The handle must be locked before accessing the data referenced by it and should be unlocked as soon as possible. Locked handle data cannot be purged or relocated. The handle is locked with a call to `mpeos_memLockH()` specifying a non-NULL *address* argument. The handle is unlocked with a call to `mpeos_memLockH()` specifying a NULL *address* argument. Each call to lock the *handle* should have a matching call to unlock the *handle*.

related function(s) `mpeos_memAllocH`
`mpeos_memCompact`
`mpeos_memFreeH`
`mpeos_memPurge`
`mpeos_memReallocH`

mpeos_memPurge

attempts to purge memory.

syntax `mpe_Error mpeos_memPurge(mpe_MemColor color, uint32_t priority);`

parameter(s) `color` specifies the type or intended use of the memory, depending on the implementation. `mpe_MemColor` is defined in the `mpe_MemColor` section.

`priority` the priority of purge candidate handles. Currently, the following values are supported for `priority`:

`MPE_MEM_PRIOR_HIGH`
indicates the highest purgeable priority.

`MPE_MEM_PRIOR_LOW`
indicates the lowest purgeable priority.

value returned If the call is successful, `mpeos_memPurge()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE ENOMEM` indicates the function failed due to insufficient memory resources.

description `mpeos_memPurge()` should attempt to purge handle-based allocations of the given or lower priority. If `priority` is not supported, then purging only takes place if `MPE_MEM_PRIOR_HIGH` is specified. `mpeos_memPurge()` will not purge unpurgeable or locked data. If the implementation does not support purging, `mpeos_memPurge()` does nothing except return `MPE_SUCCESS`.

related function(s) `mpeos_memAllocH`
`mpeos_memFreeH`
`mpeos_memReallocH`

mpeos_memReallocH

resizes the block of memory via a handle.

syntax `mpe_Error mpeos_memReallocH(`
`mpe_MemColor color,`
`uint32_t size,`
`uint32_t priority,`
`mpe_MemHandle * handle);`

| | | |
|----------------------------|---|--|
| parameter(s) | <i>color</i> | specifies the type or intended use of the memory, depending on the implementation. <code>mpe_MemColor</code> is defined in the <i>mpe_MemColor</i> section. |
| | <i>size</i> | specifies the new size, in bytes, of the memory block. |
| | <i>priority</i> | is the new priority for the handle. Purgeable handles with lower priority are more likely to be purged than higher. Currently, the following values are supported for <i>priority</i> : |
| | <code>MPE_MEM_NOPURGE</code> | indicates an unpurgeable priority. |
| | <code>MPE_MEM_PRIOR_HIGH</code> | indicates the highest purgeable priority. |
| | <code>MPE_MEM_PRIOR_LOW</code> | indicates the lowest purgeable priority. |
| | <i>handle</i> | is an input/output pointer to the memory block to resize. <code>mpe_MemHandle</code> is defined in the <i>mpe_MemHandle</i> section. If <i>handle</i> is passed an invalid pointer, the call is equivalent to <code>mpeos_memAllocH()</code> . |
| value returned | If the call is successful, <code>mpeos_memReallocH()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return one of the following error codes: | |
| | <code>MPE EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| | <code>MPE ENOMEM</code> | indicates the function failed due to insufficient memory resources. |
| description | <code>mpeos_memReallocH()</code> should resize the memory block specified by the <i>handle</i> . If necessary, a new block is allocated and the contents of the old block copied. This can also be used to change the priority of a handle. | |
| related function(s) | mpeos_memAllocH mpeos_memCompact mpeos_memFreeH mpeos_memLockH mpeos_memPurge | |

mpeos_memReallocPGen

resizes the block of memory via a pointer.

syntax `mpe_Error mpeos_memReallocPGen(`
`mpe_MemColor color,`
`uint32_t size,`
`void ** memory);`

parameter(s) `color` specifies the type or intended use of the memory, depending on the implementation. `mpe_MemColor` is defined in the `mpe_MemColor` section.

`size` specifies the new size, in bytes, of the memory block. If `size` is equal to 0, the call is equivalent to `mpeos_memFreeP()`.

`memory` is an input pointer to the pointer to the memory block to be resized and is used to return the address of the new memory block on the successful completion. If `memory` is NULL, `mpeos_memReallocPGen()` has the same effect as a call to `mpeos_memAllocP`.

value returned If the call is successful, `mpeos_memReallocPGen()` should return `MPE_SUCCESS`. Otherwise, it should not free the memory referred to by `memory` and return one of the following error codes:

`MPE EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE ENOMEM` indicates the function failed due to insufficient memory resources.

description `mpeos_memReallocPGen()` should resize, in bytes, the memory block via a pointer. The contents are unchanged to the minimum of the old and new sizes and any newly allocated memory is uninitialized. The routine conforms to POSIX behavior.

The `mpeos_memReallocP` macro, as specified by the definition of `MPE_FEATURE_MEM_PROF`, maps to `mpeos_memReallocPGen()` when profiling is disabled. `mpeos_memReallocP` is described in the `mpeos_memReallocP` section.

related function(s) `mpeos_memAllocPGen`
`mpeos_memFreePGen`

mpeos_memReallocPProf

resizes the memory block for testing memory functions.

syntax `mpe_Error mpeos_memReallocPProf(mpe_MemColor color, uint32_t size, void **memory, char *fileName, uint32_t lineNumber);`

| | | |
|---------------------|-----------------|---|
| parameter(s) | <i>color</i> | specifies the type or intended use of the memory, depending on the implementation. <code>mpe_MemColor</code> is defined in the <i>mpe_MemColor</i> section. |
| | <i>size</i> | specifies the size, in bytes, of the memory block to allocate. |
| | <i>memory</i> | is an input pointer to the pointer to the memory block to be resized and is used to return the address of the new memory block on the successful completion. If <i>memory</i> is NULL, the call is equivalent to <code>mpeos_memAllocPProf()</code> . |
| | <i>fileName</i> | is an input pointer to the file name. |
| | <i>lineNum</i> | specifies the line number in the file. |

value returned If the call is successful, `mpeos_memReallocPProf()` should return `MPE_SUCCESS`, along with a pointer to the new memory block specified by *memory*. Otherwise, it should return one of the following error codes:

| | |
|-------------------------|--|
| <code>MPE EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| <code>MPE ENOMEM</code> | indicates the function failed due to insufficient memory resources. |

description `mpeos_memReallocPProf()` should resize a block of system memory of the specified by *size*. The address of the memory block allocated is returned via the *memory* pointer. `mpeos_memReallocPProf()` is used for profiling (testing) versions of the memory functions.

The `mpeos_memReallocP` macro, as specified by the definition of `MPE_FEATURE_MEM_PROF`, maps to `mpeos_memReallocPProf()` when profiling is enabled. `mpeos_memReallocP` is described in the *mpeos_memReallocP* section.

- ◆ **NOTE:** `mpeos_memReallocPProf()` should not be used in production code, because there is run-time and memory overhead associated with the memory profiling.

related function(s) `mpeos_memFreePProf`
`mpeos_memAllocPProf`

mpeos_memRegisterMemFreeCallback

registers a function to call in case of memory exhaustion.

syntax

```
mpe_Error mpeos_memRegisterMemFreeCallback(
    mpe_MemColor color,
    int32 (*function)(mpe_MemColor, int32, int64, void *),
    void * data );
```

| | | |
|---------------------|-----------------|--|
| parameter(s) | <i>color</i> | specifies the type of memory the callback function can be expected to free. <i>mpe_MemColor</i> is defined in the <i>mpe_MemColor</i> section. |
| | <i>function</i> | is an input pointer to the callback function that takes <i>color</i> , <i>size</i> , in bytes, context identifier, and abstract data as parameters. <i>mpe_MemColor</i> identifies the color (memory) currently being requested, plus reclamation flags. <i>int32</i> identifies the amount of memory being requested. <i>int64</i> is an opaque value identifying the context in which memory is being requested. <i>void</i> identifies additional abstract context data to be passed to the callback (originally passed to <i>mpeos_memRegisterMemFreeCallback()</i> as the <i>data</i> parameter). |
| | <i>data</i> | is an input pointer to additional abstract context data to be passed to the callback function. |

| | |
|-----------------------|---|
| value returned | If the call is successful, <i>mpeos_memRegisterMemFreeCallback()</i> should return <i>MPE_SUCCESS</i> . Otherwise, it should return one of the following error codes: |
| | <i>MPE EINVAL</i> indicates at least one input parameter to the function has an invalid value. |
| | <i>MPE ENODATA</i> specifies no data is available for the specified request. |

| | |
|--------------------|---|
| description | <i>mpeos_memRegisterMemFreeCallback()</i> should register a callback which should be called when memory is low and the memory manager needs the various components to release memory. When memory is low, the memory manager should request that the components free memory, and should specify the amount of memory it wants. The callback function should attempt to free the memory, and, when finished, return the amount of memory it released. Currently, the following values may be returned: |
| 0 | indicates the callback was unable to free any memory. |
| -1 | indicates an unknown amount of memory was reclaimed and that more may be possible. |
| <i>non-zero</i> | indicates the exact amount of memory successfully reclaimed and that more may be possible. |

- ◆ **NOTE:** Additional possible values for *color* (*MEM_MEM_CALLBACK_SYSEVENT* and *MPE_MEM_CALLBACK_KILLAPP*) are defined to support the registration of Monitor Application notification, via Resource Depletion Event and application destruction callbacks. These are only meant to be referenced by the appropriate JNI for registration of the corresponding callbacks.

related function(s)

`mpeos_memAllocH`
`mpeos_memAllocPGen`
`mpeos_memUnregisterMemFreeCallback`

mpeos_memStats

outputs the current internal memory statistics.

syntax void mpeos_memStats(
 mpe_Bool *toConsole*,
 mpe_MemColor *color*,
 const char * *label*);

| | | |
|----------------------------|------------------|--|
| parameter(s) | <i>toConsole</i> | specifies where the statistics are output. If <i>toConsole</i> is TRUE, the statistics should be output to stdout. If <i>toConsole</i> is FALSE, the statistics should be output to the path specified in the .ini file specified by MPE_MEMPATH. <i>mpe_Bool</i> is defined in the <i>mpe_Bool</i> section. |
| | <i>color</i> | specifies the type of memory statistics to output. <i>mpe_MemColor</i> is defined in the <i>mpe_MemColor</i> section. To output statistics for all types of memory, specify <i>color</i> as MPE_MEM_SYSTEM. |
| | <i>label</i> | is an input pointer to a string to preface the statistical output. Specifying NULL omits the preface. |
| value returned | none | |
| description | mpeos_memStats() | should force an output of the current internal memory statistics, if any are available. |
| related function(s) | none | |

mpeos_memUnregisterMemFreeCallback

unregisters a callback to be called when memory is low.

syntax `mpe_Error mpeos_memUnregisterMemFreeCallback(mpe_MemColor color, int32 (*function)(mpe_MemColor, int32, int64, void *), void * data);`

| | | |
|---------------------|-----------------|--|
| parameter(s) | <i>color</i> | specifies the type of memory the callback function can be expected to unregistered. <code>mpe_MemColor</code> is defined in the <code>mpe_MemColor</code> section. |
| | <i>function</i> | is an input pointer to the callback function that takes color, size, in bytes, context identifier, and abstract data as parameters. All parameters should be identical to a previous call to <code>mpeos_memRegisterMemFreeCallback()</code> . |
| | <i>data</i> | is an input pointer to additional abstract context data to be passed to the callback function. |

value returned If the call is successful, `mpeos_memUnregisterMemFreeCallback()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

| | |
|--------------------------|--|
| <code>MPE EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| <code>MPE ENODATA</code> | specifies no data is available for the specified request. |

description `mpeos_memUnregisterMemFreeCallback()` should unregister a callback, originally registered with `mpeos_memRegisterMemFreeCallback()` to be called when memory is low. The unregister parameters must match the parameters in the current registration. This function is used in case a memory allocation fails.

related function(s) `mpeos_memAllocH`
`mpeos_memAllocPGen`
`mpeos_memRegisterMemFreeCallback`

MOD Module Support API

Overview

The module support Application Programming Interface (API) provides support for dynamically linked libraries modules, which are required to support the Java Virtual Machine (JVM) and useful in supporting functional separation within the OpenCable Application Platform (OCAP) stack implementation. The dynamic module support API supplies the functionality for locating library modules at runtime, lookup of symbolic values within modules (that is, function and variable names), and termination of use of modules on completion if necessary.

Before reading this chapter, you should be:

- ◆ familiar with the semantics of the module support APIs
- ◆ familiar with the concepts, implementation, and use of dynamic libraries
- ◆ familiar with the use of libraries in supporting java system libraries containing JNI support

After reading this chapter, you should be able to port the module functionality within the OCAP stack

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the module support functions:

MPE_EINVAL
MPE_SUCCESS

MPE_NODATA

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

MPE_EINVAL

MPE_EINVAL indicates at least one input parameter to the function has an invalid value. MPE_EINVAL is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_NODATA

MPE_NODATA indicates the associated resource, queue, etc. has no data available. MPE_NODATA is defined as follows:

```
#define MPE_NODATA OS_NODATA
```

MPE_SUCCESS

MPE_SUCCESS indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). MPE_SUCCESS is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types, which are defined in `mpeos_d11.h`, `mpe_error.h`, and `mpe_types.h`, are used by the module support functions:

| | |
|---------------------------|---|
| <code>mpe_Dlmod</code> | <code>mpe_DlmodData</code> |
| <code>mpe_Error</code> | <code>os_Dlmod</code> and <code>os_DlmodData</code> |
| <code>os_SymbolTbl</code> | <code>symbolDesc</code> |
| <code>_symbolDesc</code> | |

`mpe_Dlmod`

`mpe_Dlmod` specifies the dynamically-linked module identifier/handle
`mpe_Dlmod` is used to identify a previously opened/linked module.
`mpe_Dlmod` is defined in `mpeos_d11.h` as follows:

```
typedef os_Dlmod mpe_Dlmod;
```

`mpe_DlmodData`

`mpe_DlmodData` specifies the dynamically-linked data structure type used to allocate the underlying implementation structure for `mpe_Dlmod`.
`mpe_DlmodData` is defined in `mpeos_d11.h` as follows:

```
typedef os_DlmodData mpe_DlmodData;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`os_Dlmod` and `os_DlmodData`

`os_Dlmod` specifies a handle to an `os_DlmodData` structure. `os_DlmodData` specifies the DLL handle structure. `os_Dlmod` and `os_DlmodData` are defined in `os_d11.h` as follows:

```
typedef struct _os_DLMod {
    void *d11_handle;
    os_SymbolTbl *d11_symbols;
} *os_Dlmod, os_DlmodData;
```

where

`d11_handle` specifies the operating system, module handle.

`d11_symbols` is a pointer to the module symbol table.

`os_SymbolTbl`

`os_SymbolTbl` specifies symbol table contained in each library module.
`os_SymbolTbl` is defined in `os_d11.h` as follows:

```
typedef struct {
    uint32_t ht_size;
    const symbolDesc *ht_table;
} os_SymbolTbl;
```

where

`ht_size` specifies the size of the symbol hash-table.

`ht_table` is a pointer to the symbol hash-table.

symbolDesc

symbolDesc specifies symbol table contained in each library module.
symbolDesc is defined in *os_d11.h* as follows:

```
typedef struct _symbolDesc symbolDesc;
```

_symbolDesc

_symbolDesc specifies symbol table contained in each library module.
_symbolDesc is defined in *os_d11.h* as follows:

```
typedef struct _symbolDesc {  
    int32 hash;  
    char* name;  
    void* value;  
    symbolDesc* next;  
} symbolDesc_s;
```

where

| | |
|--------------|---|
| <i>hash</i> | specifies the symbol has value. |
| <i>name</i> | is a pointer to the symbol name. |
| <i>value</i> | is a pointer to the symbol value. |
| <i>next</i> | is a pointer to the next symbol in line (for example, colliding symbols). |

Supported functions

The following module support functions need to be supported:

`mpeos_d1modClose`
terminates the module.

`mpeos_d1modGetSymbol`
locates symbol information for the target symbol.

`mpeos_d1modInit`
initializes support to the global function table.

`mpeos_d1modOpen`
loads and initializes the specified module.

mpeos_dlmodClose

terminates the module.

syntax mpe_Error mpeos_dlmodClose(mpe_Dlmod *dlmodId*);

parameter(s) *dlmodId* specifies the identifier of the target module to close. *dlmodId* is returned by a previous call to mpeos_dlmodOpen().
mpe_Dlmod is defined in the *mpe_Dlmod* section.

value returned If the call is successful, mpeos_dlmodClose() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *dlmodId* has an invalid value.

description mpeos_dlmodClose() should terminate the use of the module specified by *dlmodId*.

related function(s) mpeos_dlmodGetSymbol
mpeos_dlmodOpen

mpeos_dlmodGetSymbol

locates symbol information for the target symbol.

syntax mpe_Error mpeos_dlmodGetSymbol(
 mpe_Dlmod *dlmodId*,
 const char * *symbol*,
 void ** *value*);

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>dlmodId</i> | specifies the identifier of the target module to search for the specified symbol. <i>dlmodId</i> is returned by a previous call to mpeos_dlmodOpen(). <i>mpe_Dlmod</i> is defined in the <i>mpe_Dlmod</i> section. |
| | <i>symbol</i> | is an input pointer to the name of the target symbol in which to search. |
| | <i>value</i> | is an output pointer for returning the associated value of the target symbol. |

value returned If the call is successful, mpeos_dlmodGetSymbol() should return MPE_SUCCESS and a pointer to the value of the target symbol (*value*). Otherwise, it should return one of the following error codes:

- MPE_EINVAL indicates at least one input parameter to the function has an invalid value.
- MPE_ENODATA indicates *symbol* is not valid.

description mpeos_dlmodGetSymbol() should be used for locating a target symbol within a module. Specifically, mpeos_dlmodGetSymbol() returns a pointer (*value*) to the symbol information for a specific symbol within a specific module.

related function(s) mpeos_dlmodClose
 mpeos_dlmodOpen

mpeos_dlmmodInit

initializes support to the global function table.

syntax void mpeos_dlmmodInit(void ** *mpe_ftable*);

parameter(s) *mpe_ftable* is an input pointer to the MPE global function table.

value returned none

description *mpeos_dlmmodInit()* is called only by MPE and only during initialization; therefore, *mpeos_dlmmodInit()* is not a general purpose function. *mpeos_dlmmodInit()* should initialize the MPEOS DLL support with the MPE global function table (*mpe_ftable*). This populates the porting layer with the MPE global function table pointer, so that it can be passed to library modules during initialization of the modules.

related function(s) none

mpeos_dlmodOpen

loads and initializes the specified module.

syntax `mpe_Error mpeos_dlmodOpen(const char * name, mpe_Dlmod * dlmodId);`

| | | |
|---------------------|----------------------|--|
| parameter(s) | <code>name</code> | is an input pointer to the name of the dynamic module to open. |
| | <code>dlmodId</code> | is an output pointer for returning the identifier of the opened module. <code>mpe_Dlmod</code> is defined in the <i>mpe_Dlmod</i> section. |

value returned If the call is successful, `mpeos_dlmodOpen()` should return `MPE_SUCCESS` and a pointer to the opened module (`dlmodId`). Otherwise, it should return the following error code:

`MPE_EINVAL` indicates `name` has an invalid value.

description `mpeos_dlmodOpen()` should load, locate, link, and initialize the module specified by `name`. The module may be loaded from a file system or located in RAM or ROM. This initialization interface returns a ID/handle for association of the module and subsequent symbol lookup.

related function(s) `mpeos_dlmodClose`
`mpeos_dlmodGetSymbol`

NET Networking API

Overview

The networking Application Programming Interface (API) provides a consistent interface to Berkeley System Distribution/Portable Operating System Interface for Computer Environments (BSD/POSIX) style sockets regardless of the underlying operating system. This isolates the Java Virtual Machine (JVM) and Java applications from differences between platforms.

The networking API is syntactically and semantically very close to the definition of BSD/POSIX sockets. Therefore, for most platforms this API will map directly to the underlying operating system API. For platforms without BSD/POSIX style sockets this API provides the proper translation to networking support in the operating system.

Support for both IPv4 and IPv6 is included. However, IPv6 is not mandatory.

Before reading this chapter, you should be:

- ◆ familiar with networking standards for Transmission Control Protocol/Internet Protocol (TCP/IP) and Domain Name Server (DNS)
- ◆ familiar with the networking APIs provided by BSD/POSIX sockets
- ◆ familiar with the OpenCable Application Platform (OCAP) networking packages including `java.net`, `javax.net` and `org.dvb.net`

After reading this chapter, you should be:

- ◆ able to port the networking functionality within the OCAP stack
- ◆ knowledgeable on run-time requirements your port must fulfill

Definitions

The networking Application Programming Interface (API) needs definitions in the following categories:

- ◆ *Protocol-independent definitions*
- ◆ *IPv4-specific definitions*
- ◆ *IPv6-specific definitions*
- ◆ *Socket-level socket definition options*
- ◆ *IPv4-level socket definition options*
- ◆ *IPv6-level socket definition options*
- ◆ *TCP-level socket definition options*

Protocol-independent definitions

The following definitions, which are defined in `mpeos_socket.h`, are used by the event functions for network support:

| | |
|--|--|
| <code>MPE_SOCKET_DGRAM</code> | <code>MPE_SOCKET_FD_SETSIZE</code> |
| <code>MPE_SOCKET_FIONBIO</code> | <code>MPE_SOCKET_FIONREAD</code> |
| <code>MPE_SOCKET_INVALID_SOCKET</code> | <code>MPE_SOCKET_MAXHOSTNAMELEN</code> |
| <code>MPE_SOCKET_MSG_OOB</code> | <code>MPE_SOCKET_MSG_PEEK</code> |
| <code>MPE_SOCKET_SHUT_RD</code> | <code>MPE_SOCKET_SHUT_RDWR</code> |
| <code>MPE_SOCKET_SHUT_WR</code> | <code>MPE_SOCKET_STREAM</code> |

`MPE_SOCKET_DGRAM`

`MPE_SOCKET_DGRAM` specifies a datagram-based socket.

`MPE_SOCKET_DGRAM` is defined as follows:

```
#define MPE_SOCKET_DGRAM OS_SOCKET_DGRAM
```

`MPE_SOCKET_FD_SETSIZE`

`MPE_SOCKET_FD_SETSIZE` specifies the maximum number of file descriptors represented in the `mpe_SocketFDSet` data type. `MPE_SOCKET_FD_SETSIZE` is defined as follows:

```
#define MPE_SOCKET_FD_SETSIZE OS_SOCKET_FD_SETSIZE
```

`MPE_SOCKET_FIONBIO`

`MPE_SOCKET_FIONBIO` is used with `mpeos_socketIoctl()` to clear or turn on the non-blocking flag for the socket. `MPE_SOCKET_FIONBIO` is defined as follows:

```
#define MPE_SOCKET_FIONBIO OS_SOCKET_FIONBIO
```

MPE_SOCKET_FIONREAD

MPE_SOCKET_FIONREAD is used with `mpeos_socketIoctl()` to return the number of bytes currently in the receive buffer for the socket.

MPE_SOCKET_FIONREAD is defined as follows:

```
#define MPE_SOCKET_FIONREAD OS_SOCKET_FIONREAD
```

MPE_SOCKET_INVALID_SOCKET

MPE_SOCKET_INVALID_SOCKET specifies a value which indicates an `mpe_Socket` descriptor is not valid. MPE_SOCKET_INVALID_SOCKET is defined as follows:

```
#define MPE_SOCKET_INVALID_SOCKET OS_SOCKET_INVALID_SOCKET
```

MPE_SOCKET_MAXHOSTNAMELEN

MPE_SOCKET_MAXHOSTNAMELEN specifies the maximum length of a host name in bytes, including the terminating NULL byte. MPE_SOCKET_MAXHOSTNAMELEN is defined as follows:

```
#define MPE_SOCKET_MAXHOSTNAMELEN OS_SOCKET_MAXHOSTNAMELEN
```

MPE_SOCKET_MSG_OOB

MPE_SOCKET_MSG_OOB sends or receives out-of-band data on sockets that support out-of-band data. The significance and semantics of the out-of-band data are protocol-specific. MPE_SOCKET_MSG_OOB is defined as follows:

```
#define MPE_SOCKET_MSG_OOB OS_SOCKET_MSG_OOB
```

MPE_SOCKET_MSG_PEEK

MPE_SOCKET_MSG_PEEK peeks at an incoming message. The data is treated as unread and the next receive operation should still return this data.

MPE_SOCKET_MSG_PEEK is defined as follows:

```
#define MPE_SOCKET_MSG_PEEK OS_SOCKET_MSG_PEEK
```

MPE_SOCKET_SHUT_RD

MPE_SOCKET_SHUT_RD is used with `mpeos_socketShutdown()` to disable further receive operations. MPE_SOCKET_SHUT_RD is defined as follows:

```
#define MPE_SOCKET_SHUT_RD OS_SOCKET_SHUTDOWN_RD
```

MPE_SOCKET_SHUT_RDWR

MPE_SOCKET_SHUT_RDWR is used with `mpeos_socketShutdown()` to disable further send and receive operations. MPE_SOCKET_SHUT_RDWR is defined as follows:

```
#define MPE_SOCKET_SHUT_RDWR OS_SOCKET_SHUTDOWN_RDWR
```

MPE_SOCKET_SHUT_WR

MPE_SOCKET_SHUT_WR is used with `mpeos_socketShutdown()` to disable further send operations. MPE_SOCKET_SHUT_WR is defined as follows:

```
#define MPE_SOCKET_SHUT_WR OS_SOCKET_SHUTDOWN_WR
```

MPE_SOCKET_STREAM

MPE_SOCKET_STREAM specifies a stream-based socket. MPE_SOCKET_STREAM is defined as follows:

```
#define MPE_SOCKET_STREAM OS_SOCKET_STREAM
```

IPv4-specific definitions

The following IPv4-specific definitions, which are defined in `mpeos_socket.h`, are used by the event functions for networking support:

| | |
|--|--|
| <code>MPE_SOCKET_AF_INET4</code> | <code>MPE_SOCKET_IN4ADDR_ANY</code> |
| <code>MPE_SOCKET_IN4ADDR_LOOPBACK</code> | <code>MPE_SOCKET_INET4_ADDRSTRLEN</code> |
| <code>MPE_SOCKET IPPROTO_IPV4</code> | <code>MPE_SOCKET IPPROTO_TCP</code> |
| <code>MPE_SOCKET IPPROTO_UDP</code> | |

`MPE_SOCKET_AF_INET4`

`MPE_SOCKET_AF_INET4` specifies the address family for IPv4 sockets.
`MPE_SOCKET_AF_INET4` is defined as follows:

```
#define MPE_SOCKET_AF_INET4 OS_SOCKET_AF_INET4
```

`MPE_SOCKET_IN4ADDR_ANY`

`MPE_SOCKET_IN4ADDR_ANY` specifies the wildcard IPv4 address (matches any address). `MPE_SOCKET_IN4ADDR_ANY` is defined as follows:

```
#define MPE_SOCKET_IN4ADDR_ANY OS_SOCKET_IN4ADDR_ANY
```

`MPE_SOCKET_INET4_ADDRSTRLEN`

`MPE_SOCKET_INET4_ADDRSTRLEN` specifies the maximum length of an IPv4 address string (includes the null terminator).

`MPE_SOCKET_INET4_ADDRSTRLEN` is defined as follows:

```
#define MPE_SOCKET_INET4_ADDRSTRLEN OS_SOCKET_INET4_ADDRSTRLEN
```

`MPE_SOCKET_IN4ADDR_LOOPBACK`

`MPE_SOCKET_IN4ADDR_LOOPBACK` specifies the IPv4 loopback address.
`MPE_SOCKET_IN4ADDR_LOOPBACK` is defined as follows:

```
#define MPE_SOCKET_IN4ADDR_LOOPBACK OS_SOCKET_IN4ADDR_LOOPBACK
```

`MPE_SOCKET IPPROTO_IPV4`

`MPE_SOCKET IPPROTO_IPV4` specifies the IPv4 protocol.
`MPE_SOCKET IPPROTO_IPV4` is defined as follows:

```
#define MPE_SOCKET IPPROTO_IPV4 OS_SOCKET IPPROTO_IPV4
```

`MPE_SOCKET IPPROTO_TCP`

`MPE_SOCKET IPPROTO_TCP` specifies the Transmission Control Protocol/Internet Protocol (TCP/IP). `MPE_SOCKET IPPROTO_TCP` is defined as follows:

```
#define MPE_SOCKET IPPROTO_TCP OS_SOCKET IPPROTO_TCP
```

MPE_SOCKET IPPROTO_UDP

MPE_SOCKET IPPROTO_UDP specifies the User Datagram Protocol/Internet Protocol (UDP/IP). MPE_SOCKET IPPROTO_UDP is defined as follows:

```
#define MPE_SOCKET IPPROTO_UDP OS_SOCKET IPPROTO_UDP
```

IPv6-specific definitions

The following IPv6-specific definitions, which are defined in `mpeos_socket.h`, are used by the event functions for networking support:

| | |
|---|--|
| <code>MPE_SOCKET_AF_INET6</code> | <code>MPE_SOCKET_IN6ADDR_ANY_INIT</code> |
| <code>MPE_SOCKET_IN6ADDR_LOOPBACK_INIT</code> | |
| <code>MPE_SOCKET_INET6_ADDRSTRLEN</code> | <code>MPE_SOCKET IPPROTO_IPV6</code> |

`MPE_SOCKET_IN6ADDR_ANY_INIT`

`MPE_SOCKET_IN6ADDR_ANY_INIT` specifies a wildcard IPv6 address (that is, it matches any address). `MPE_SOCKET_IN6ADDR_ANY_INIT` is defined as follows:

```
#define MPE_SOCKET_IN6ADDR_ANY_INIT OS_SOCKET_IN6ADDR_ANY_INIT
```

`MPE_SOCKET_IN6ADDR_LOOPBACK_INIT`

`MPE_SOCKET_IN6ADDR_LOOPBACK_INIT` specifies the IPv6 loopback address. `MPE_SOCKET_IN6ADDR_LOOPBACK_INIT` is defined as follows:

```
#define MPE_SOCKET_IN6ADDR_LOOPBACK_INIT  
OS_SOCKET_IN6ADDR_LOOPBACK_INIT
```

`MPE_SOCKET_INET6_ADDRSTRLEN`

`MPE_SOCKET_INET6_ADDRSTRLEN` specifies the maximum length of an IPv6 address string. `MPE_SOCKET_INET6_ADDRSTRLEN` is defined as follows:

```
#define MPE_SOCKET_INET6_ADDRSTRLEN OS_SOCKET_INET6_ADDRSTRLEN
```

`MPE_SOCKET_AF_INET6`

`MPE_SOCKET_AF_INET6` specifies the address family for IPv6 sockets. This constant is defined only when `MPE_FEATURE_IPV6` is defined.

`MPE_SOCKET_AF_INET6` is defined as follows:

```
#define MPE_SOCKET_AF_INET6 OS_SOCKET_AF_INET6
```

`MPE_SOCKET IPPROTO_IPV6`

`MPE_SOCKET IPPROTO_IPV6` specifies the IPv6 protocol.

`MPE_SOCKET IPPROTO_IPV6` is defined as follows:

```
#define MPE_SOCKET IPPROTO_IPV6 OS_SOCKET IPPROTO_IPV6
```

Socket-level socket definition options

The following definitions, which are defined in `mpeos_socket.h`, are used by the event functions for optional networking support:

| | |
|--------------------------------------|-------------------------------------|
| <code>MPE_SOCKET_SO_BROADCAST</code> | <code>MPE_SOCKET_SO_DEBUG</code> |
| <code>MPE_SOCKET_SO_DONTROUTE</code> | <code>MPE_SOCKET_SO_ERROR</code> |
| <code>MPE_SOCKET_SO_KEEPALIVE</code> | <code>MPE_SOCKET_SO_LINGER</code> |
| <code>MPE_SOCKET_SO_OOBINLINE</code> | <code>MPE_SOCKET_SO_RCVBUF</code> |
| <code>MPE_SOCKET_SO_RCVLOWAT</code> | <code>MPE_SOCKET_SO_RCVTIMEO</code> |
| <code>MPE_SOCKET_SO_REUSEADDR</code> | <code>MPE_SOCKET_SO_SNDBUF</code> |
| <code>MPE_SOCKET_SO SNDLOWAT</code> | <code>MPE_SOCKET_SO_SNDSIZEO</code> |
| <code>MPE_SOCKET_SO_TYPE</code> | <code>MPE_SOCKET_SO_SOCKET</code> |

`MPE_SOCKET_SO_BROADCAST`

`MPE_SOCKET_SO_BROADCAST` controls whether transmission of broadcast messages is supported by the protocol. This Boolean option should store an `int` value. `MPE_SOCKET_SO_BROADCAST` is defined as follows:

```
#define MPE_SOCKET_SO_BROADCAST OS_SOCKET_SO_BROADCAST
```

`MPE_SOCKET_SO_DEBUG`

`MPE_SOCKET_SO_DEBUG` controls whether debugging information is being recorded. This Boolean option should store an `int` value.

`MPE_SOCKET_SO_DEBUG` is defined as follows:

```
#define MPE_SOCKET_SO_DEBUG OS_SOCKET_SO_DEBUG
```

`MPE_SOCKET_SO_DONTROUTE`

`MPE_SOCKET_SO_DONTROUTE` controls whether outgoing messages bypass the standard routing facilities. The destination should be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of `MPE_SOCKET_SO_DONTROUTE` depends on what protocol is in use. This Boolean option should store an `int` value. `MPE_SOCKET_SO_DONTROUTE` is defined as follows:

```
#define MPE_SOCKET_SO_DONTROUTE OS_SOCKET_SO_DONTROUTE
```

`MPE_SOCKET_SO_ERROR`

`MPE_SOCKET_SO_ERROR` reports information about error status and clears it when used with `mpeos_socketGetOpt()`. This option cannot be set. This option should store an `int` value. `MPE_SOCKET_SO_ERROR` is defined as follows:

```
#define MPE_SOCKET_SO_ERROR OS_SOCKET_SO_ERROR
```

MPE_SOCKET_SO_KEEPALIVE

MPE_SOCKET_SO_KEEPALIVE controls whether connections are kept active with periodic transmission of messages, if the protocol supports this. If the connected socket fails to respond to these messages, the connection should be broken and threads writing to that socket should be notified with an MPE_SIGPIPE signal. This Boolean option should store an `int` value.

MPE_SOCKET_SO_KEEPALIVE is defined as follows:

```
#define MPE_SOCKET_SO_KEEPALIVE OS_SOCKET_SO_KEEPALIVE
```

MPE_SOCKET_SO_LINGER

MPE_SOCKET_SO_LINGER controls whether the socket lingers on `mpeos_socketClose()` if data is present. If the MPE_SOCKET_SO_LINGER option is set, the system blocks the process during `mpeos_socketClose()` until all the data can be transmitted or until the end of the interval indicated by `mpe_SocketLinger`, whichever comes first. If the MPE_SOCKET_SO_LINGER option is not specified and `mpeos_socketClose()` is issued, the system handles the call in a way that allows the process to continue as quickly as possible. The MPE_SOCKET_SO_LINGER option should store an `mpeos_Linger` structure. MPE_SOCKET_SO_LINGER is defined as follows:

```
#define MPE_SOCKET_SO_LINGER OS_SOCKET_SO_LINGER
```

MPE_SOCKET_SO_OOBINLINE

MPE_SOCKET_SO_OOBINLINE controls whether the socket leaves received out-of-band data (data marked urgent) inline. This Boolean option should store an `int` value. MPE_SOCKET_SO_OOBINLINE is defined as follows:

```
#define MPE_SOCKET_SO_OOBINLINE OS_SOCKET_SO_OOBINLINE
```

MPE_SOCKET_SO_RCVBUF

MPE_SOCKET_SO_RCVBUF controls the size of the receive buffer. This option should store an `int` value. MPE_SOCKET_SO_RCVBUF is defined as follows:

```
#define MPE_SOCKET_SO_RCVBUF OS_SOCKET_SO_RCVBUF
```

MPE_SOCKET_SO_RCVLOWAT

MPE_SOCKET_SO_RCVLOWAT controls the minimum number of bytes (or low-water mark value) to process for socket input operations. The default value for the option is 1. If the option is set to a larger value, blocking receive calls normally wait until they have received the minimum number of bytes or the requested amount. (The blocking receive calls may return less than the minimum number of bytes if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data should store an `int` value.) MPE_SOCKET_SO_RCVLOWAT is defined as follows:

```
#define MPE_SOCKET_SO_RCVLOWAT OS_SOCKET_SO_RCVLOWAT
```

MPE_SOCKET_SO_RCVTIMEO

MPE_SOCKET_SO_RCVTIMEO controls the time-out value for input operations. The MPE_SOCKET_SO_RCVTIMEO option should store an `mpe_TimeVal` structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, a partial count or an error indication, if no data was received, should return. In the latter case, the error code can be retrieved with `mpeos_socketGetLastError()`.

The default for MPE_SOCKET_SO_RCVTIMEO is zero, which indicates a receive operation should not time out. The option should store an `mpe_TimeVal` structure. MPE_SOCKET_SO_RCVTIMEO is defined as follows:

```
#define MPE_SOCKET_SO_RCVTIMEO OS_SOCKET_SO_RCVTIMEO
```

MPE_SOCKET_SO_REUSEADDR

MPE_SOCKET_SO_REUSEADDR controls whether the rules used in validating addresses supplied to `mpeos_socketBind()` should allow reuse of local addresses, if the protocol supports this. This Boolean option should store an `int` value. MPE_SOCKET_SO_REUSEADDR is defined as follows:

```
#define MPE_SOCKET_SO_REUSEADDR OS_SOCKET_SO_REUSEADDR
```

MPE_SOCKET_SO_SNDBUF

MPE_SOCKET_SO_SNDBUF controls the send buffer size . This option should store an `int` value. MPE_SOCKET_SO_SNDBUF is defined as follows:

```
#define MPE_SOCKET_SO_SNDBUF OS_SOCKET_SO_SNDBUF
```

MPE_SOCKET_SO SNDLOWAT

MPE_SOCKET_SO SNDLOWAT controls the minimum number of bytes (or low-water mark value) to process for socket output operations. Non-blocking output operations should process no data if flow control does not allow the smaller of the send minimum number of bytes or the entire request to be processed. This option should store an `int` value. MPE_SOCKET_SO SNDLOWAT is defined as follows:

```
#define MPE_SOCKET_SO SNDLOWAT OS_SOCKET_SO SNDLOWAT
```

MPE_SOCKET_SO SNDTIMEO

MPE_SOCKET_SO SNDTIMEO controls the time-out value specifying the amount of time an output function blocks due to flow control preventing data from being sent. If a send operation has blocked for the specified amount of time, a partial count or an error indication, if no data was sent, should return. In the latter case, the error code can be retrieved with `mpeos_socketGetLastError()`. The default for MPE_SOCKET_SO SNDTIMEO is zero, which indicates a send operation should not time out. The option should store an `mpe_TimeVal` structure. MPE_SOCKET_SO SNDTIMEO is defined as follows:

```
#define MPE_SOCKET_SO SNDTIMEO OS_SOCKET_SO SNDTIMEO
```

MPE_SOCKET_SO_TYPE

MPE_SOCKET_SO_TYPE reports the socket type when used with `mpeos_socketGetOpt()`. You cannot set the MPE_SOCKET_SO_TYPE option. MPE_SOCKET_SO_TYPE should store an int value. MPE_SOCKET_SO_TYPE is defined as follows:

```
#define MPE_SOCKET_SO_TYPE OS_SOCKET_SO_TYPE
```

MPE_SOCKET_SOL_SOCKET

MPE_SOCKET_SOL_SOCKET is used with `mpeos_socketSetOpt()` and `mpeos_socketGetOpt()` to specify a socket level option. MPE_SOCKET_SOL_SOCKET is defined as follows:

```
#define MPE_SOCKET_SOL_SOCKET OS_SOCKET_SOL_SOCKET
```

IPv4-level socket definition options

The following IPv4-level definitions, which are defined in `mpeos_socket.h`, are used by the event functions for optional networking support:

```
MPE_SOCKET_IPV4_ADD_MEMBERSHIP
MPE_SOCKET_IPV4_DROP_MEMBERSHIP MPE_SOCKET_IPV4_MULTICAST_IF
MPE_SOCKET_IPV4_MULTICAST_LOOP MPE_SOCKET_IPV4_MULTICAST_TTL
```

`MPE_SOCKET_IPV4_ADD_MEMBERSHIP`

`MPE_SOCKET_IPV4_ADD_MEMBERSHIP` joins a multicast group on a specified local interface. The argument is an `mpe_SocketIPv4McastReq` structure. `imr_multiaddr` contains the address of the multicast group the application wants to join or leave. `imr_multiaddr` must be a valid multicast address. `imr_interface` is the address of the local interface with which the system should join the multicast group; if `imr_interface` is equal to `MPE_SOCKET_IN4ADDR_ANY`, the system chooses an appropriate interface.

More than one join is allowed on a given socket, but each join must be for a different multicast address, or for the same multicast address but on a different interface from previous joins for that address on this socket. This can be used on a multihomed host where, for example, one socket is created and then for each interface a join is performed for a given multicast address. `MPE_SOCKET_IPV4_ADD_MEMBERSHIP` is defined as follows:

```
#define MPE_SOCKET_IPV4_ADD_MEMBERSHIP
OS_SOCKET_IPV4_ADD_MEMBERSHIP
```

`MPE_SOCKET_IPV4_DROP_MEMBERSHIP`

`MPE_SOCKET_IPV4_DROP_MEMBERSHIP` leaves a multicast group. The argument is an `mpe_SocketIPv4McastReq` structure similar to `MPE_SOCKET_IPV4_ADD_MEMBERSHIP`. If the local interface is not specified (that is, the value is `INADDR_ANY`), the first matching multicasting group membership is dropped.

If a process joins a group but never explicitly leaves the group, when the socket is closed (either explicitly or on process termination), the membership is dropped automatically. It is possible for multiple processes on a host to each join the same group, in which case the host remains a member of that group until the last process leaves the group. `MPE_SOCKET_IPV4_DROP_MEMBERSHIP` is defined as follows:

```
#define MPE_SOCKET_IPV4_DROP_MEMBERSHIP
OS_SOCKET_IPV4_DROP_MEMBERSHIP
```

MPE_SOCKET IPV4 MULTICAST IF

MPE_SOCKET_IPV4_MULTICAST_IF specifies the interface for outgoing multicast datagrams sent on this socket. This interface is specified as an mpe_SocketIPv4Addr structure. If the value specified is INADDR_ANY, any interface previously assigned by this socket is removed and the system will choose the interface each time a datagram is sent.

Be careful to distinguish between the local interface specified (or chosen) when a process joins a group (the interface on which arriving multicast datagrams are received), and the local interface specified (or chosen) when a multicast datagram is output. MPE_SOCKET_IPV4_MULTICAST_IF is defined as follows:

```
#define MPE_SOCKET_IPV4_MULTICAST_IF  
OS_SOCKET_IPV4_MULTICAST_IF
```

MPE_SOCKET IPV4 MULTICAST LOOP

MPE_SOCKET_IPV4_MULTICAST_LOOP enables or disables the local loopback of multicast datagrams. By default, loopback is enabled. A copy of each multicast datagram sent by a process on the host will also be looped back and processed as a received datagram by that host, if the host belongs to that multicast group on the outgoing interface. This option should store an unsigned char value. MPE_SOCKET_IPV4_MULTICAST_LOOP is defined as follows:

```
#define MPE_SOCKET_IPV4_MULTICAST_LOOP  
OS_SOCKET_IPV4_MULTICAST_LOOP
```

MPE_SOCKET IPV4 MULTICAST TTL

MPE_SOCKET_IPV4_MULTICAST_TTL sets or reads the time-to-live value of outgoing multicast datagrams for this socket. It is essential for multicast packets to set the smallest time-to-live possible. The default is 1, which means the multicast packets do not leave the local network unless the user program explicitly requests this. This option should store an unsigned char value. MPE_SOCKET_IPV4_MULTICAST_TTL is defined as follows:

```
#define MPE_SOCKET_IPV4_MULTICAST_TTL  
OS_SOCKET_IPV4_MULTICAST_TTL
```

IPv6-level socket definition options

The following IPv6-level definitions, which are defined in `mpeos_socket.h`, are used by the event functions for optional networking support:

```
MPE_SOCKET_IPV6_ADD_MEMBERSHIP
MPE_SOCKET_IPV6_DROP_MEMBERSHIP
MPE_SOCKET_IPV6_MULTICAST_HOPS  MPE_SOCKET_IPV6_MULTICAST_IF
MPE_SOCKET_IPV6_MULTICAST_LOOP
```

`MPE_SOCKET_IPV6_ADD_MEMBERSHIP`

`MPE_SOCKET_IPV6_ADD_MEMBERSHIP` joins a multicast group on a specified local interface. The argument is an `mpe_SocketIPv6McastReq` structure. `ipv6mr_multiaddr` contains the address of the multicast group the application wants to join or leave. `ipv6mr_multiaddr` must be a valid multicast address. `ipv6mr_interface` is the address of the local interface with which the system should join the multicast group; if `ipv6mr_interface` is equal to `MPE_SOCKET_IN6ADDR_ANY_INIT`, the system chooses an appropriate interface.

More than one join is allowed on a given socket, but each join must be for a different multicast address, or for the same multicast address but on a different interface from previous joins for that address on this socket. This can be used on a multihomed host where, for example, one socket is created and then for each interface a join is performed for a given multicast address. `MPE_SOCKET_IPV6_ADD_MEMBERSHIP` is defined as follows:

```
#define MPE_SOCKET_IPV6_ADD_MEMBERSHIP
OS_SOCKET_IPV6_ADD_MEMBERSHIP
```

`MPE_SOCKET_IPV6_DROP_MEMBERSHIP`

`MPE_SOCKET_IPV6_DROP_MEMBERSHIP` leaves a multicast group. The argument is a `mpe_SocketIPv6McastReq` structure similar to `MPE_SOCKET_IPV6_ADD_MEMBERSHIP`. If the local interface is not specified (that is, the value is `IN6ADDR_ANY_INIT`), the first matching multicasting group membership is dropped.

If a process joins a group but never explicitly leaves the group, when the socket is closed (either explicitly or on process termination), the membership is dropped automatically. It is possible for multiple processes on a host to each join the same group, in which case the host remains a member of that group until the last process leaves the group.

`MPE_SOCKET_IPV6_DROP_MEMBERSHIP` is defined as follows:

```
#define MPE_SOCKET_IPV6_DROP_MEMBERSHIP
OS_SOCKET_IPV6_DROP_MEMBERSHIP
```

MPE_SOCKET_IPV6_MULTICAST_HOPS

MPE_SOCKET_IPV6_MULTICAST_HOPS sets or reads the hop limit of outgoing multicast datagrams for this socket. It is essential for multicast packets to set the smallest hop limit possible. The default is 1, which means multicast packets do not leave the local network unless the user program explicitly requests this. This option should store an `unsigned int` value.

MPE_SOCKET_IPV6_MULTICAST_HOPS is defined as follows:

```
#define MPE_SOCKET_IPV6_MULTICAST_HOPS  
OS_SOCKET_IPV6_MULTICAST_HOPS
```

MPE_SOCKET_IPV6_MULTICAST_IF

MPE_SOCKET_IPV6_MULTICAST_IF specifies the interface for outgoing multicast datagrams sent on this socket. This interface is specified as a `mpe_SocketIPv6Addr` structure. If the value specified is `IN6ADDR_ANY_INIT`, any interface previously assigned by this socket option is removed, and the system will choose the interface each time a datagram is sent.

Be careful to distinguish between the local interface specified (or chosen) when a process joins a group (the interface on which arriving multicast datagrams are received), and the local interface specified (or chosen) when a multicast datagram is output. MPE_SOCKET_IPV6_MULTICAST_IF is defined as follows:

```
#define MPE_SOCKET_IPV6_MULTICAST_IF  
OS_SOCKET_IPV6_MULTICAST_IF
```

MPE_SOCKET_IPV6_MULTICAST_LOOP

MPE_SOCKET_IPV6_MULTICAST_LOOP enables or disables the local loopback of multicast datagrams. By default, the loopback is enabled. A copy of each multicast datagram sent by a process on the host will also be looped back and processed as a received datagram by that host, if the host belongs to that multicast group on the outgoing interface. This option should store an `unsigned int` value. MPE_SOCKET_IPV6_MULTICAST_LOOP is defined as follows:

```
#define MPE_SOCKET_IPV6_MULTICAST_LOOP  
OS_SOCKET_IPV6_MULTICAST_LOOP
```

TCP-level socket definition options

The following TCP-level definitions, which are defined in `mpeos_socket.h`, are used by the event functions for optional networking support:

```
MPE_SOCKET_TCP_NODELAY
```

```
MPE_SOCKET_TCP_NODELAY
```

`MPE_SOCKET_TCP_NODELAY` disables TCP's Nagle algorithm, when set. By default, the algorithm is enabled. This option should store an `int` value.

```
#define MPE_SOCKET_TCP_NODELAY OS_SOCKET_TCP_NODELAY
```

Error codes

The following error codes, which are defined in `mpeos_socket.h` and `mpe_error.h`, are used by the event functions for the networking Application Programming Interface (API):

| | |
|--------------------------|----------------------------|
| MPE_SOCKET_EACCES | MPE_SOCKET_EADDRINUSE |
| MPE_SOCKET_EADDRNOTAVAIL | MPE_SOCKET_EAFNOSUPPORT |
| MPE_SOCKET_EAGAIN | MPE_SOCKET_EALREADY |
| MPE_SOCKET_EBADF | MPE_SOCKET_ECONNABORTED |
| MPE_SOCKET_ECONNREFUSED | MPE_SOCKET_ECONNRESET |
| MPE_SOCKET_EDESTADDRREQ | MPE_SOCKET_EDOM |
| MPE_SOCKET_EHOSTNOTFOUND | MPE_SOCKET_EHOSTUNREACH |
| MPE_SOCKET_EINTR | MPE_SOCKET_EIO |
| MPE_SOCKET_EISCONN | MPE_SOCKET_ELOOP |
| MPE_SOCKET_EMFILE | MPE_SOCKET_EMSGSIZE |
| MPE_SOCKET_ENAMETOOLONG | MPE_SOCKET_ENETDOWN |
| MPE_SOCKET_ENETUNREACH | MPE_SOCKET_ENFILE |
| MPE_SOCKET_ENOBUFS | MPE_SOCKET_ENOPROTOOPT |
| MPE_SOCKET_ENORECOVERY | MPE_SOCKET_ENOSPC |
| MPE_SOCKET_ENOTCONN | MPE_SOCKET_ENOTSOCK |
| MPE_SOCKET_EOPNOTSUPP | MPE_SOCKET_EPIPE |
| MPE_SOCKET_EPROTO | MPE_SOCKET_EPROTONOSUPPORT |
| MPE_SOCKET_EPROTOTYPE | MPE_SOCKET_ETIMEDOUT |
| MPE_SOCKET_ETIME | MPE_SOCKET_EWOULDBLOCK |

The following error codes, which are defined in `mpe_error.h`, are used by the event functions for the networking API:

| | |
|------------|------------------|
| MPE_EINVAL | MPE_ENODATA |
| MPE_ENOMEM | MPE_ETHREADDEATH |

MPE_EINVAL

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined in `mpeos_error.h` as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENODATA

`MPE_ENODATA` specifies the server recognized the request and the name, but no address is available. Another type of request to the name server for the domain might return an answer. `MPE_ENODATA` is defined in `mpeos_error.h` as follows:

```
#define MPE_ENODATA OS_ENODATA
```

MPE_ENOMEM

`MPE_ENOMEM` indicates the function failed due to insufficient memory resources. `MPE_ENOMEM` is defined in `mpeos_error.h` as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_ETHREADDEATH

`MPE_ETHREADDEATH` indicates the thread executing the function has been marked for death. `MPE_ETHREADDEATH` is defined in `mpeos_error.h` as follows:

```
#define MPE_ETHREADDEATH OS_ETHREADDEATH
```

MPE_SOCKET_EACCES

MPE_SOCKET_EACCES indicates access was denied due to insufficient privileges. MPE_SOCKET_EACCES is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EACCES OS_MPE_SOCKET_EACCES
```

MPE_SOCKET_EADDRINUSE

MPE_SOCKET_EADDRINUSE indicates the specified address is already in use. MPE_SOCKET_EADDRINUSE is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EADDRINUSE OS_MPE_SOCKET_EADDRINUSE
```

MPE_SOCKET_EADDRNOTAVAIL

MPE_SOCKET_EADDRNOTAVAIL indicates the specified address is not available from the local machine. MPE_SOCKET_EADDRNOTAVAIL is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EADDRNOTAVAIL OS_MPE_SOCKET_EADDRNOTAVAIL
```

MPE_SOCKET_EAFNOSUPPORT

MPE_SOCKET_EAFNOSUPPORT indicates the specified address is not supported. MPE_SOCKET_EAFNOSUPPORT is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EAFNOSUPPORT OS_MPE_SOCKET_EAFNOSUPPORT
```

MPE_SOCKET_EAGAIN

MPE_SOCKET_EAGAIN specifies the operation timed out and should be tried again. MPE_SOCKET_EAGAIN is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EAGAIN OS_MPE_SOCKET_EAGAIN
```

MPE_SOCKET_EALREADY

MPE_SOCKET_EALREADY indicates a connection request is already in progress for the specified socket. MPE_SOCKET_EALREADY is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EALREADY OS_MPE_SOCKET_EALREADY
```

MPE_SOCKET_EBADF

MPE_SOCKET_EBADF indicates an invalid file descriptor. MPE_SOCKET_EBADF is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EBADF OS_MPE_SOCKET_EBADF
```

MPE_SOCKET_ECONNABORTED

MPE_SOCKET_ECONNABORTED indicates a connection has been aborted. MPE_SOCKET_ECONNABORTED is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ECONNABORTED OS_MPE_SOCKET_ECONNABORTED
```

MPE_SOCKET_ECONNREFUSED

MPE_SOCKET_ECONNREFUSED indicates the connection request was refused.
MPE_SOCKET_ECONNREFUSED is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ECONNREFUSED OS_MPE_SOCKET_ECONNREFUSED
```

MPE_SOCKET_ECONNRESET

MPE_SOCKET_ECONNRESET indicates the connection was reset.
MPE_SOCKET_ECONNRESET is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ECONNRESET OS_MPE_SOCKET_ECONNRESET
```

MPE_SOCKET_EDESTADDRREQ

MPE_SOCKET_EDESTADDRREQ indicates an invalid destination address.
MPE_SOCKET_EDESTADDRREQ is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EDESTADDRREQ OS_MPE_SOCKET_EDESTADDRREQ
```

MPE_SOCKET_EDOM MPE_SOCKET_EDOM indicates an invalid field value. MPE_SOCKET_EDOM is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EDOM OS_MPE_SOCKET_EDOM
```

MPE_SOCKET_EHOSTNOTFOUND

MPE_SOCKET_EHOSTNOTFOUND indicates the host is unknown.
MPE_SOCKET_EHOSTNOTFOUND is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EHOSTNOTFOUND OS_MPE_SOCKET_EHOSTNOTFOUND
```

MPE_SOCKET_EHOSTUNREACH

MPE_SOCKET_EHOSTUNREACH indicates the host cannot be reached.
MPE_SOCKET_EHOSTUNREACH is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EHOSTUNREACH OS_MPE_SOCKET_EHOSTUNREACH
```

MPE_SOCKET_EINTR MPE_SOCKET_EINTR indicates a signal interrupted the function before it could complete the requested operation. MPE_SOCKET_EINTR is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EINTR OS_MPE_SOCKET_EINTR
```

MPE_SOCKET_EIO MPE_SOCKET_EIO indicates an input/output (I/O) error occurred.
MPE_SOCKET_EIO is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EIO OS_MPE_SOCKET_EIO
```

MPE_SOCKET_EISCONN

MPE_SOCKET_EISCONN indicates the socket is already connected.
MPE_SOCKET_EISCONN is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EISCONN OS_MPE_SOCKET_EISCONN
```

MPE_SOCKET_ELOOP

MPE_SOCKET_ELOOP indicates more than the maximum number of loops occurred. MPE_SOCKET_ELOOP is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ELOOP OS_MPE_SOCKET_ELOOP
```

MPE_SOCKET_EMFILE

MPE_SOCKET_EMFILE indicates no more file descriptors are available for this process. MPE_SOCKET_EMFILE is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EMFILE OS_MPE_SOCKET_EMFILE
```

MPE_SOCKET_EMSGSIZE

MPE_SOCKET_EMSGSIZE indicates the maximum message size was exceeded. MPE_SOCKET_EMSGSIZE is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EMSGSIZE OS_MPE_SOCKET_EMSGSIZE
```

MPE_SOCKET_ENAMETOOLONG

MPE_SOCKET_ENAMETOOLONG indicates the maximum pathname length was exceeded. MPE_SOCKET_ENAMETOOLONG is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENAMETOOLONG OS_MPE_SOCKET_ENAMETOOLONG
```

MPE_SOCKET_ENETDOWN

MPE_SOCKET_ENETDOWN indicates the local network interface used to reach the destination is down. MPE_SOCKET_ENETDOWN is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENETDOWN OS_MPE_SOCKET_ENETDOWN
```

MPE_SOCKET_ENETUNREACH

MPE_SOCKET_ENETUNREACH indicates no route to the network is present. MPE_SOCKET_ENETUNREACH is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENETUNREACH OS_MPE_SOCKET_ENETUNREACH
```

MPE_SOCKET_ENFILE

MPE_SOCKET_ENFILE indicates no more file descriptors are available for the system. MPE_SOCKET_ENFILE is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENFILE OS_MPE_SOCKET_ENFILE
```

MPE_SOCKET_ENOBUFS

MPE_SOCKET_ENOBUFS indicates the function failed due to insufficient available resources in the system. MPE_SOCKET_ENOBUFS is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENOBUFS OS_MPE_SOCKET_ENOBUFS
```

MPE_SOCKET_ENOPROTOOPT

MPE_SOCKET_ENOPROTOOPT indicates the option is not supported by the protocol. MPE_SOCKET_ENOPROTOOPT is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENOPROTOOPT OS_MPE_SOCKET_ENOPROTOOPT
```

MPE_SOCKET_ENORECOVERY

MPE_SOCKET_ENORECOVERY indicates an unexpected server failure occurred and cannot be recovered. MPE_SOCKET_ENORECOVERY is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENORECOVERY OS_MPE_SOCKET_ENORECOVERY
```

MPE_SOCKET_ENOSPC

MPE_SOCKET_ENOSPC indicates the size of the result buffer is inadequate. MPE_SOCKET_ENOSPC is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENOSPC OS_MPE_SOCKET_ENOSPC
```

MPE_SOCKET_ENOTCONN

MPE_SOCKET_ENOTCONN indicates the socket is not connected. MPE_SOCKET_ENOTCONN is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENOTCONN OS_MPE_SOCKET_ENOTCONN
```

MPE_SOCKET_ENOTSOCK

MPE_SOCKET_ENOTSOCK indicates that a parameter does not refer to a socket. MPE_SOCKET_ENOTSOCK is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ENOTSOCK OS_MPE_SOCKET_ENOTSOCK
```

MPE_SOCKET_EOPNOTSUPP

MPE_SOCKET_EOPNOTSUPP indicates an unsupported operation. MPE_SOCKET_EOPNOTSUPP is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EOPNOTSUPP OS_MPE_SOCKET_EOPNOTSUPP
```

MPE_SOCKET_EPIPE

MPE_SOCKET_EPIPE indicates an invalid socket or that the socket is no longer connected. MPE_SOCKET_EPIPE is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EPIPE OS_MPE_SOCKET_EPIPE
```

MPE_SOCKET_EPROTO

MPE_SOCKET_EPROTO indicates an invalid protocol. MPE_SOCKET_EPROTO is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EPROTO OS_MPE_SOCKET_EPROTO
```

MPE_SOCKET_EPROTONOSUPPORT

MPE_SOCKET_EPROTONOSUPPORT indicates an unsupported protocol. MPE_SOCKET_EPROTONOSUPPORT is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EPROTONOSUPPORT  
OS_MPE_SOCKET_EPROTONOSUPPORT
```

MPE_SOCKET_EPROTOTYPE

MPE_EPROTONOTYPE indicates an incompatible protocol for the socket type. MPE_EPROTONOTYPE is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EPROTOTYPE OS_MPE_SOCKET_EPROTOTYPE
```

MPE_SOCKET_ETIMEDOUT

MPE_SOCKET_ETIMEDOUT indicates a time-out occurred during the requested operation. MPE_SOCKET_ETIMEDOUT is defined in `mpeos_error.h` as follows:

```
#define MPEETIMEOUT OSETIMEOUT
```

MPE_SOCKET_ETIMEDOUT is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ETIMEDOUT OS_MPE_SOCKET_ETIMEDOUT
```

MPE_SOCKET_ETRYAGAIN

MPE_SOCKET_ETRYAGAIN indicates a temporary and possibly transient error occurred, such as a failure of a server to respond. MPE_SOCKET_ETRYAGAIN is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_ETRYAGAIN OS_MPE_SOCKET_ETRYAGAIN
```

MPE_SOCKET_EWOULDBLOCK

MPE_SOCKET_EWOULDBLOCK indicates the operation would block if attempted. MPE_SOCKET_EWOULDBLOCK is defined in `mpeos_socket.h` as follows:

```
#define MPE_SOCKET_EWOULDBLOCK OS_MPE_SOCKET_EWOULDBLOCK
```

Data types and structures

The networking Application Programming Interface (API) needs definitions and structures in the following categories:

- ◆ General data types
- ◆ IPv4 data types
- ◆ IPv6 data types

General data types

The following data types, which are defined in `mpeos_socket.h` and `mpeos_time.h`, are used by the network functions:

| | |
|---------------------------------|----------------------------------|
| <code>mpe_Bool</code> | <code>mpe_Socket</code> |
| <code>mpe_SocketFDSet</code> | <code>mpe_SocketHostEntry</code> |
| <code>mpe_SocketLinger</code> | <code>mpe_SocketSaFamily</code> |
| <code>mpe_SocketSockAddr</code> | <code>mpe_SocketSockLen</code> |
| <code>mpe_TimeVal</code> | |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Socket`

`mpe_Socket` is the numeric type used to reference an open socket file descriptor. `mpe_Socket` is defined in `mpeos_socket.h` as follows:

```
typedef os_Socket mpe_Socket;
```

`mpe_SocketFDSet`

`mpe_SocketFDSet` is an opaque structure used by `mpeos_socketSelect()` to define a set of file descriptors. `mpe_SocketFDSet` is defined in `mpeos_socket.h` as follows:

```
typedef os_SocketFDSet mpe_SocketFDSet;
```

`mpe_SocketHostEntry`

`mpe_SocketHostEntry` is the host entry returned by calls to `mpeos_socketGetHostByAddr()` and `mpeos_socketGetHostByName()`. `mpe_SocketHostEntry` is defined in `mpeos_socket.h` as follows:

```
typedef os_SocketHostEntry mpe_SocketHostEntry;
```

The following members of `mpe_SocketHostEntry` may be accessed directly and must be present in the underlying platform-dependent structure:

`char *h_name`

specifies the official (canonical) name of host.

`char **h_aliases`

is a pointer to an array of pointers to the alias names.

`int h_addrtype` specifies the host address type.

`int h_length` specifies the length of the address.

`char ** h_addr_list`

is a pointer to an array of pointers with IPv4 or IPv6 addresses.

mpe_SocketLinger `mpe_SocketLinger` specifies a toggle. If `l_onoff` is 0, toggling is off. If `l_onoff` is not zero, toggling is on. `mpe_SocketLinger` is defined in `mpeos_socket.h` as follows:

```
typedef os_SocketLinger mpe_SocketLinger;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

| | |
|---------------------------|--|
| <code>int l_onoff</code> | specifies a toggle, where 0 = off and non-zero = on. |
| <code>int l_linger</code> | specifies the linger time in seconds. |

mpe_SocketSaFamily

`mpe_SocketSaFamily` is the socket address family (that is, MPE_SOCKET_AF_INET4 or MPE_SOCKET_AF_INET6). `mpe_SocketSaFamily` is defined in `mpeos_socket.h` as follows:

```
typedef os_SocketSaFamily mpe_SocketSaFamily;
```

mpe_SocketSockAddr

`mpe_SocketSockAddr` is the protocol-independent socket address structure. `mpe_SocketSockAddr` is defined in `mpeos_socket.h` as follows:

```
typedef os_SocketSockAddr mpe_SocketSockAddr;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

| | |
|---|-------------------------------|
| <code>mpe_SocketSaFamily sa_family</code> | specifies the address family. |
|---|-------------------------------|

mpe_SocketSockLen

`mpe_SocketSockLen` defines a length of a socket structure. `mpe_SocketSockLen` is defined in `mpeos_socket.h` as follows:

```
typedef os_SocketSockLen mpe_SocketSockLen;
```

mpe_TimeVal

`mpe_TimeVal` is a time representation value commonly used in supporting the BSD style Multimedia Platform Environment Operating System (MPEOS) Socket API. `mpe_TimeVal` is defined in `mpeos_time.h` as follows:

```
typedef os_TimeVal mpe_TimeVal;
```

IPv4 data types

The following data types, which are defined in `mpeos_socket.h`, are used by the network functions:

| | |
|-------------------------------------|-------------------------------------|
| <code>mpe_SocketIPv4Addr</code> | <code>mpe_SocketIPv4McastReq</code> |
| <code>mpe_SocketIPv4SockAddr</code> | |

`mpe_SocketIPv4Addr`

`mpe_SocketIPv4Addr` holds the IPv4 address. `mpe_SocketIPv4Addr` is defined as follows:

```
typedef os_SocketIPv4Addr mpe_SocketIPv4Addr;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

`uint32_t s_addr` specifies the 32-bit IPv4 address in network byte order.

`mpe_SocketIPv4McastReq`

`mpe_SocketIPv4McastReq` is the IPv4 multicast request structure.

`mpe_SocketIPv4McastReq` is defined as follows:

```
typedef os_SocketIPv4McastReq mpe_SocketIPv4McastReq;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

`mpe_SocketIPv4Addr imr_multiaddr`
specifies the IPv4 class D multicast address.

`mpe_SocketIPv4Addr imr_interface`
specifies the IPv4 address of the local interface.

`mpe_SocketIPv4SockAddr`

`mpe_SocketIPv4SockAddr` is the IPv4 socket address structure.

`mpe_SocketIPv4SockAddr` is defined as follows:

```
typedef os_SocketIPv4SockAddr mpe_SocketIPv4SockAddr;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

`mpe_SocketSaFamily sin_family`
specifies the address family (MPE_SOCKET_AF_INET4).

`uint16_t sin_port`
specifies the 16-bit Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port number.

`mpe_SocketIPv4Addr sin_addr`
specifies the IPv4 address.

IPv6 data types

The following data types, which are defined in `mpeos_socket.h`, are used by the network functions:

| | |
|-------------------------------------|-------------------------------------|
| <code>mpe_SocketIPv6Addr</code> | <code>mpe_SocketIPv6McastReq</code> |
| <code>mpe_SocketIPv6SockAddr</code> | |

`mpe_SocketIPv6Addr`

`mpe_SocketIPv6Addr` specifies the IPv6 address. This structure is defined only when `MPE_FEATURE_IPV6` is defined. `mpe_SocketIPv6Addr` is defined as follows:

```
typedef os_SocketIPv6Addr mpe_SocketIPv6Addr;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

| | |
|----------------------------------|---|
| <code>uint8_t s6_addr[16]</code> | specifies the 128-bit IPv6 address in network byte order. |
|----------------------------------|---|

`mpe_SocketIPv6McastReq`

`mpe_SocketIPv6McastReq` specifies the IPv6 multicast request structure. This structure is defined only when `MPE_FEATURE_IPV6` is defined.

`mpe_SocketIPv6McastReq` is defined as follows:

```
typedef os_SocketIPv6McastReq mpe_SocketIPv6McastReq;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

| | |
|--|---------------------------------------|
| <code>mpe_SocketIPv6Addr ipv6mr_multiaddr</code> | specifies the IPv6 multicast address. |
|--|---------------------------------------|

| | |
|--|--------------------------------------|
| <code>mpe_SocketIPv6Addr ipv6mr_interface</code> | specifies the interface index, or 0. |
|--|--------------------------------------|

`mpe_SocketIPv6SockAddr`

`mpe_SocketIPv6SockAddr` specifies the IPv6 socket address structure. This structure is defined only when `MPE_FEATURE_IPV6` is defined.

`mpe_SocketIPv6SockAddr` is defined as follows:

```
typedef os_SocketIPv6SockAddr mpe_SocketIPv6SockAddr;
```

The following members may be accessed directly and must be present in the underlying platform-dependent structure:

| | |
|---|--|
| <code>mpe_SocketSaFamily sin6_family</code> | specifies the address family (<code>MPE_SOCKET_AF_INET6</code>). |
|---|--|

| | |
|---------------------------------|--|
| <code>uint16_t sin6_port</code> | specifies the transport layer port number in network byte order. |
|---------------------------------|--|

| | |
|-------------------------------------|--|
| <code>uint32_t sin6_flowinfo</code> | specifies the priority and flow label in network byte order. |
|-------------------------------------|--|

| | |
|---|-----------------------------|
| <code>mpe_SocketIPv6Addr sin6_addr</code> | specifies the IPv6 address. |
|---|-----------------------------|

Supported functions

The following networking functions need to be supported:

- `mpeos_socketAccept`
accepts a connection.
- `mpeos_socketAtоН`
interprets the specified character string.
- `mpeos_socketBind`
binds a socket address name to a specified socket.
- `mpeos_socketClose`
deallocates the specified file descriptor.
- `mpeos_socketConnect`
attempts to make a connection to a socket.
- `mpeos_socketCreate`
creates an unbound socket in a communications domain.
- `mpeos_socketFDClear`
removes the file descriptor from the specified set.
- `mpeos_socketFDIsSet`
determines whether the file descriptor is a part of the specified set.
- `mpeos_socketFDSet`
adds a file descriptor to a specified set.
- `mpeos_socketFDZero`
initializes the specified descriptor set to the NULL set.
- `mpeos_socketGetHostByAddr`
gets an entry containing addresses.
- `mpeos_socketGetHostByName`
gets an entry containing addresses for a specified host.
- `mpeos_socketGetHostName`
gets the standard host name for the current machine.
- `mpeos_socketGetLastError`
gets the error status of the last socket function.
- `mpeos_socketGetOpt`
manipulates options associated with a socket.
- `mpeos_socketGetPeerName`
gets the peer address of the specified socket.
- `mpeos_socketGetSockName`
gets the locally-bound name of the specified socket.
- `mpeos_socketHtoNL`
converts a 32-bit quantity from host byte order to network byte order.

`mpeos_socketHtoNS`
converts a 16-bit quantity from host byte order to network byte order.

`mpeos_socketInit`
initializes the networking API.

`mpeos_socketIoctl`
performs a variety of control functions on a socket.

`mpeos_socketListen`
marks a socket as accepting connections.

`mpeos_socketNtoA`
converts an Internet address to an ASCII string.

`mpeos_socketNtoHL`
converts a 32-bit quantity from network byte order to host byte order.

`mpeos_socketNtoHS`
converts a 16-bit quantity from network byte order to host byte order.

`mpeos_socketNtoP`
converts a numeric address into a text string.

`mpeos_socketPtoN`
converts a text string to a numeric address.

`mpeos_socketRecv`
receives a message from a socket.

`mpeos_socketRecvFrom`
receives a message and retrieves the source address from a socket.

`mpeos_socketSelect`
examines file descriptor sets.

`mpeos_socketSend`
initiates sending a message from a socket to the socket's peer.

`mpeos_socketSendTo`
sends a message through a connection-mode or connectionless-mode socket.

`mpeos_socketSetOpt`
sets the specified option.

`mpeos_socketShutdown`
shuts down all or part of a full-duplex connection.

`mpeos_socketTerm`
terminates the networking API.

mpeos_socketAccept

accepts a connection.

syntax `mpe_Socket mpeos_socketAccept(mpe_Socket socket, mpe_SocketSockAddr * address, mpe_SocketSockLen * address_len);`

| | | |
|---------------------|--------------------|--|
| parameter(s) | <i>socket</i> | specifies the socket that was created with <code>mpeos_socketCreate()</code> , has been bound to an address with <code>mpeos_socketBind()</code> , and has issued a successful call to <code>mpeos_socketListen()</code> . <code>mpe_Socket</code> is described in the <i>mpe_Socket</i> section. |
| | <i>address</i> | is either a NULL pointer or an output pointer to the <code>mpe_SocketSockAddr</code> structure where the address of the connecting socket should be returned. <code>mpe_SocketSockAddr</code> is described in the <i>mpe_SocketSockAddr</i> section. |
| | <i>address_len</i> | is an output pointer to a length parameter specifying the length of the supplied <code>mpe_SocketSockAddr</code> structure when <code>mpeos_socketAccept()</code> is called. When the call returns, <i>address_len</i> should specify the length of the stored address. <code>mpe_SocketSockLen</code> is described in the <i>mpe_SocketSockLen</i> section. |

value returned `mpeos_socketAccept()` should return the non-negative file descriptor of the accepted socket. Otherwise, `mpeos_socketAccept()` should return `MPE_SOCKET_INVALID_SOCKET` and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

| | |
|--------------------------------------|--|
| <code>MPE_SOCKET_EBADF</code> | indicates an invalid file descriptor. |
| <code>MPE_SOCKET_ECONNABORTED</code> | indicates a connection has been aborted. |
| <code>MPE_EINVAL</code> | specifies either the <i>socket</i> has been shut down, bound to an address, or the protocol does not support binding to a new address. |
| <code>MPE_SOCKET_EMFILE</code> | indicates no more file descriptors are available for this process. |
| <code>MPE_SOCKET_ENFILE</code> | indicates no more file descriptors are available for the system. |
| <code>MPE_SOCKET_ENOBUFS</code> | indicates the function failed due to insufficient available resources in the system. |

| | |
|---------------------|--|
| MPE_ENOMEM | indicates the function failed due to insufficient memory resources. |
| MPE_SOCKET_ENOTSOCK | indicates that a parameter does not refer to a socket. |
| MPE_ETHREADDEATH | indicates the thread executing the function has been marked for death. |

description mpeos_socketAccept() should do the following:

1. Extract the first connection on the queue of pending connections.
2. Create a new socket with the same socket type protocol and address family as the specified *socket*.
3. Allocate a new file descriptor for that socket.

If *address* is not a NULL pointer, the address of the peer for the accepted connection should be stored in the *mpe_SocketSockAddr* structure pointed to by *address*, and the length of this address should be stored in the object pointed to by *address_len*.

If the actual length of the address is greater than the length of the supplied *mpe_SocketSockAddr* structure, mpeos_socketAccept() should truncate the stored address.

If the protocol permits connections by unbound clients and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

If the listen queue is empty of connection requests, mpeos_socketAccept() should block until a connection is present.

The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

related function(s)

[mpeos_socketBind](#)
[mpeos_socketCreate](#)
[mpeos_socketGetLastError](#)
[mpeos_socketListen](#)

mpeos_socketAtoN

interprets the specified character string.

syntax `int mpeos_socketAtoN(
 const char * strptr,
 mpe_SocketIPv4Addr * addrptr);`

parameter(s) `strptr` is an input pointer to an Internet address in dot notation.
`addrptr` is an input pointer to the structure to hold the address in numeric form. `mpe_SocketIPv4Addr` is described in the `mpe_SocketIPv4Addr` section.

value returned If the string is successfully interpreted, `mpeos_socketAtoN()` should return 1. Otherwise, `mpeos_socketAtoN()` should return 0 if the string is invalid.

description `mpeos_socketAtoN()` should interpret the specified character string as an Internet address, placing the address into the structure provided. All Internet addresses should be returned in network byte order (bytes are ordered from left to right). All network numbers and local address parts should be returned as machine-format integer values. Using the dot notation, specify addresses in one of the following forms:

`a.b.c.d` When four parts are specified, each part should be interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

`a.b.c` When a three-part address is specified, the last part should be interpreted as a 16-bit quantity and placed in the right-most two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as `128.network.host`.

`a.b` When a two-part address is supplied, the last part should be interpreted as a 24-bit quantity and placed in the right-most three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as `network.host`.

`a` When only one part is given, the value should be stored directly in the network address without any byte rearrangement.

All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or hexadecimal, as specified in the International Organization for Standardization (ISO) C standard (that is, a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

related function(s) `mpeos_socketNtoA`

mpeos_socketBind

binds a socket address name to a specified socket.

syntax int mpeos_socketBind(
 mpe_Socket *socket*,
 const mpe_SocketSockAddr * *address*,
 mpe_SocketSockLen *address_len*);

| | | |
|-----------------------|--|---|
| parameter(s) | <i>socket</i> | specifies the file descriptor of the socket to be bound. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>address</i> | is an input pointer to the <i>mpe_SocketSockAddr</i> structure containing the address to be bound to the socket. The length and format of the address depends on the address family of the socket. <i>mpe_SocketSockAddr</i> is described in the <i>mpe_SocketSockAddr</i> section. |
| | <i>address_len</i> | specifies the length of the <i>mpe_SocketSockAddr</i> structure pointed to by <i>address</i> . <i>mpe_SocketSockLen</i> is described in the <i>mpe_SocketSockLen</i> section. |
| value returned | If the call is successful, <i>mpeos_socketBind()</i> should return 0. Otherwise, <i>mpeos_socketBind()</i> should return -1 and by making a call to <i>mpeos_socketGetLastError()</i> one of the following error codes can be retrieved: | |
| | MPE_SOCKET_EADDRINUSE indicates the specified address is already in use. | |
| | MPE_SOCKET_EADDRNOTAVAIL indicates the specified address is not available from the local machine. | |
| | MPE_SOCKET_EBADF indicates an invalid file descriptor. | |
| | MPE EINVAL when returned for this function could have one of two meanings: specifies either the <i>socket</i> has been shut down, bound to an address, or the protocol does not support binding to a new address; or indicates at least one input parameter to the function has an invalid parameter value. | |
| | MPE_SOCKET_ENOBUFS indicates the function failed due to insufficient available resources in the system. | |
| | MPE_SOCKET_ENOTSOCK indicates that a parameter does not refer to a socket. | |

description mpeos_socketBind() should assign a local socket address to a socket specified by *socket*. Sockets created with the mpeos_socketCreate() function are initially unnamed; they are identified only by their address family.

related function(s) mpeos_socketAccept
mpeos_socketCreate
mpeos_socketGetLastError

mpeos_socketClose

deallocates the specified file descriptor.

syntax `int mpeos_socketClose(mpe_Socket socket);`

parameter(s) `socket` specifies the file descriptor associated with the socket to be closed. `mpe_Socket` is described in the *mpe_Socket* section.

value returned If the call is successful, `mpeos_socketClose()` should return 0. Otherwise, `mpeos_socketClose()` should return -1 and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

`MPE_SOCKET_EBADF`
indicates an invalid file descriptor.

`MPE_ETHREADDEATH`
indicates the thread executing the function has been marked for death.

description `mpeos_socketClose()` should deallocate the file descriptor specified by `socket`. If the file descriptor is deallocated, `mpeos_socketClose()` makes the current file descriptor available for return by subsequent calls to `mpeos_socketCreate()` or other functions that allocate file descriptors.

If the socket is in connection-mode, and the `MPE_SOCKET_SO_LINGER` option is set for the socket with non-zero linger time, and the socket has untransmitted data, then `mpeos_socketClose()` should block for up to the current linger interval until all data is transmitted.

related function(s) `mpeos_socketBind`
`mpeos_socketCreate`
`mpeos_socketGetLastError`

mpeos_socketConnect

attempts to make a connection to a socket.

syntax int mpeos_socketConnect(
 mpe_Socket *socket*,
 const mpe_SocketSockAddr * *address*,
 mpe_SocketSockLen *address_len*);

| | | |
|---------------------|--------------------|---|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket in which to make the connection. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>address</i> | is an input pointer to a <i>mpe_SocketSockAddr</i> structure containing the peer address. The length and format of the address depend on the address family of the socket. <i>mpe_SocketSockAddr</i> is described in the <i>mpe_SocketSockAddr</i> section. |
| | <i>address_len</i> | specifies the length of the <i>mpe_SocketSockAddr</i> structure pointed to by <i>address</i> . <i>mpe_SocketSockLen</i> is described in the <i>mpe_SocketSockLen</i> section. |

value returned If the call is successful, *mpeos_socketConnect()* should return 0. Otherwise, *mpeos_socketConnect()* should return -1 and by making a call to *mpeos_socketGetLastError()* one of the following error codes can be retrieved:

| | |
|--------------------------|--|
| MPE_SOCKET_EADDRINUSE | indicates the specified address is already in use. |
| MPE_SOCKET_EADDRNOTAVAIL | indicates the specified address is not available from the local machine. |
| MPE_SOCKET_EBADF | indicates an invalid file descriptor. |
| MPE_EINVAL | indicates at least one input parameter to the function has an invalid value. |
| MPE_SOCKET_ENETDOWN | indicates the local network interface used to reach the destination is down. |
| MPE_SOCKET_ENETUNREACH | indicates no route to the network is present. |
| MPE_SOCKET_ENOBUFS | indicates the function failed due to insufficient available resources in the system. |

MPE_SOCKET_ENOTSOCK

indicates that a parameter does not refer to a socket.

MPE_ETHREADDEATH

indicates the thread executing the function has been marked for death.

description

`mpeos_socketConnect()` should attempt to make a connection to a socket. If the socket has not already been bound to a local address, `mpeos_socketConnect()` should bind the socket to an unused local address.

If the initiating socket is not connection-mode, `mpeos_socketConnect()` should set the socket's peer address, and not make the connection. For MPE_SOCKET_DATAGRAM sockets, the peer address identifies where all datagrams are sent on subsequent `mpeos_socketSend()` functions, and limits the remote sender for subsequent `mpeos_socketRecv()` functions. If *address* is a NULL address for the protocol, the socket's peer address should be reset.

If the initiating socket is connection-mode, then `mpeos_socketConnect()` should attempt to establish a connection to the address specified by *address*. If the connection cannot be established immediately, `mpeos_socketConnect()` should block for up to an unspecified time-out interval until the connection is established. If the time-out interval expires before the connection is established, `mpeos_socketConnect()` should fail and the connection attempt should be aborted. If `mpeos_socketConnect()` is interrupted by a signal that is caught while blocked waiting to establish a connection, the call to `mpeos_socketConnect()` should fail, but the connection request should not be aborted, and the connection should be established asynchronously. In this case, `mpeos_socketGetLastError()` should return MPE_SOCKET_EINTR.

When the connection has been established asynchronously, `mpeos_socketSelect()` should indicate the file descriptor for the socket is ready for writing.

related function(s)

`mpeos_socketBind`
`mpeos_socketGetLastError`
`mpeos_socketRecv`
`mpeos_socketSelect`

mpeos_socketCreate

creates an unbound socket in a communications domain.

syntax `mpe_Socket mpeos_socketCreate(int domain, int type, int protocol);`

| | | |
|-----------------------|---------------------|---|
| parameter(s) | <i>domain</i> | specifies the address family used in the communications domain in which a socket is to be created. Currently, the only valid values for <i>domain</i> are MPE_SOCKET_AF_INET4 and MPE_SOCKET_AF_INET6. |
| | <i>type</i> | specifies the type of socket to be created, which determines the semantics of communication over the socket. Currently, the following values are supported for <i>type</i> : |
| | MPE_SOCKET_STREAM | specifies a stream-based socket. |
| | MPE_SOCKET_DATAGRAM | specifies a datagram-based socket. |
| | <i>protocol</i> | specifies a particular protocol to be used with the socket. If <i>protocol</i> is 0, <code>mpeos_socketCreate()</code> should use an unspecified default protocol appropriate for the requested socket type. |
| value returned | | If the call is successful, <code>mpeos_socketCreate()</code> should return a valid socket descriptor. Otherwise, <code>mpeos_socketCreate()</code> should return MPE_SOCKET_INVALID_SOCKET and by making a call to <code>mpeos_socketGetLastError()</code> one of the following error codes can be retrieved: |
| | MPE_SOCKET_EMFILE | indicates no more file descriptors are available for this process. |
| | MPE_SOCKET_ENFILE | indicates no more file descriptors are available for the system. |
| | MPE_SOCKET_ENOBUFS | indicates the function failed due to insufficient available resources in the system. |
| | MPE_ENOMEM | indicates the function failed due to insufficient memory resources. |

description mpeos_socketCreate() should create an unbound socket in a communications domain, and return a file descriptor that can be used in later function calls operating on sockets.

related function(s) mpeos_socketAccept
mpeos_socketBind
mpeos_socketClose

mpeos_socketFDClear

removes the file descriptor from the specified set.

syntax void mpeos_socketFDClear(
 mpe_Socket *fd*,
 mpe_SocketFDSet * *fdset*);

parameter(s) *fd* specifies the file descriptor to be removed from *fdset*.
mpe_Socket is described in the *mpe_Socket* section.

fdset is an input and output pointer to the file descriptor set to be modified. *mpe_SocketFDSet* is described in the *mpe_SocketFDSet* section.

value returned none

description *mpeos_socketFDClear()* should remove the file descriptor specified by *fd* from the set pointed to by *fdset*. If *fd* is not a member of this set, there should be no effect on the set and no error is returned. The behavior of *mpeos_socketFDClear()* is undefined in the following cases:

- ◆ if *fd* is less than 0
- ◆ if *fd* is greater than or equal to MPE_SOCKET_FD_SETSIZE
- ◆ if *fd* is not a valid file descriptor

related function(s) *mpeos_socketFDIsSet*
mpeos_socketFDSet
mpeos_socketFDZero

mpeos_socketFDIsSet

determines whether the file descriptor is a part of the specified set.

syntax int mpeos_socketFDIsSet(
 mpe_Socket *fd*,
 mpe_SocketFDSet * *fdset*);

parameter(s) *fd* specifies the file descriptor within *fdset* to be checked.
mpe_Socket is described in the *mpe_Socket* section.

fdset is an input pointer to the file descriptor set to be checked.
mpe_SocketFDSet is described in the *mpe_SocketFDSet* section.

value returned If the bit for the file descriptor specified by *fd* is set in the file descriptor set pointed to by *fdset*, *mpeos_socketFDIsSet()* should return a non-zero value. Otherwise, it should return 0.

description *mpeos_socketFDIsSet()* should determine whether the file descriptor *fd* is a member of the set pointed to by *fdset*. The behavior of *mpeos_socketFDIsSet()* is undefined in the following cases:

- ◆ if *fd* is less than 0
- ◆ if *fd* is greater than or equal to MPE_SOCKET_FD_SETSIZE
- ◆ if *fd* is not a valid file descriptor

related function(s) *mpeos_socketFDClear*
mpeos_socketFDSet
mpeos_socketFDZero

mpeos_socketFDSet

adds a file descriptor to a specified set.

syntax void mpeos_socketFDSet(
 mpe_Socket *fd*,
 mpe_SocketFDSet * *fdset*);

parameter(s) *fd* specifies the file descriptor to be added to *fdset*. *mpe_Socket* is described in the *mpe_Socket* section.

fdset is an input and output pointer to the file descriptor set to be modified. *mpe_SocketFDSet* is described in the *mpe_SocketFDSet* section.

value returned none

description *mpeos_socketFDSet()* should add the file descriptor *fd* to the set pointed to by *fdset*. If the file descriptor *fd* is already in this set, there should be no effect on the set, and no error is returned. The behavior of *mpeos_socketFDSet()* is undefined in the following cases:

- ◆ if *fd* is less than 0
- ◆ if *fd* is greater than or equal to MPE_SOCKET_FD_SETSIZE
- ◆ if *fd* is not a valid file descriptor

related function(s) *mpeos_socketFDClear*
mpeos_socketFDSet
mpeos_socketFDZero

mpeos_socketFDZero

initializes the specified descriptor set to the NULL set.

syntax void mpeos_socketFDZero(mpe_SocketFDSet * fdset);

parameter(s) *fdset* is an output pointer to the file descriptor set to initialize. *mpe_SocketFDSet* is described in the *mpe_SocketFDSet* section.

value returned none

description *mpeos_socketFDZero()* should initialize the descriptor set pointed to by *fdset* to the NULL set. No error is returned if the set is not empty when *mpeos_socketFDZero()* is invoked. The behavior of *mpeos_socketFDZero()* is undefined in the following cases:

- ◆ if *fd* is less than 0
- ◆ if *fd* is greater than or equal to MPE_SOCKET_FD_SETSIZE
- ◆ if *fd* is not a valid file descriptor

related function(s) *mpeos_socketFDClear*
mpeos_socketFDIsSet
mpeos_socketFDSet

mpeos_socketGetHostByAddr

gets an entry containing addresses.

syntax `mpe_SocketHostEntry * mpeos_socketGetHostByAddr(const void * addr, mpe_SocketSockLen len, int type);`

parameter(s) *addr* is an input pointer to the address, which is specified in network byte order.

len specifies the length of the address. `mpe_SocketSockLen` is described in the *mpe_SocketSockLen* section.

type specifies the type of the address.

value returned If the requested entry was found, `mpeos_socketGetHostByAddr()` should return a pointer to an `mpe_SocketHostEntry` structure.

The caller must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated, so the caller should copy any information it needs before calling any other socket function.

Otherwise, `mpeos_socketGetHostByAddr()` should return a NULL pointer, and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

`MPE_SOCKET_EHOSTNOTFOUND`

indicates the host is unknown.

`MPE_ENODATA` specifies the server recognized the request and the *name*, but no address is available. Another type of request to the name server for the domain might return an answer.

`MPE_SOCKET_ENORECOVERY`

indicates an unexpected server failure occurred and cannot be recovered.

`MPE_SOCKET_ETRYAGAIN`

indicates a temporary and possibly transient error occurred, such as a failure of a server to respond.

description `mpeos_socketGetHostByAddr()` should get an entry containing the address family type for the host with address *addr*. *len* contains the length of the address pointed to by *addr*. This information is considered to be stored in a database that can be accessed sequentially or randomly. Implementation of this database is unspecified.

addr should be an `mpe_SocketIPv4Addr` structure when *type* is `MPE_SOCKET_AF_INET4` or `mpe_SocketIPv6Addr` structure when *type* is `MPE_SOCKET_AF_INET6`. *addr* contains a binary format (that is, not NULL-terminated) address in network byte order.

`mpeos_socketGetHostByAddr()` is not guaranteed to return addresses of address families other than `MPE_SOCKET_AF_INET4` or `MPE_SOCKET_AF_INET6`, even when such addresses exist in the database.

If `mpeos_socketGetHostByAddr()` returns successfully, the `h_addrtype` field in the result should be the same as the *type* argument passed to the function, and the `h_addr_list` field should list a single address that is a copy of *addr* passed to `mpeos_socketGetHostByAddr()`.

related function(s)

`mpeos_socketGetHostByName`
`mpeos_socketGetHostName`
`mpeos_socketGetLastError`

mpeos_socketGetHostByName

gets an entry containing addresses for a specified host.

syntax `mpe_SocketHostEntry * mpeos_socketGetHostByName(const char * name);`

parameter(s) `name` is an input pointer to the name of the host to get.

value returned If the call is successful, `mpeos_socketGetHostByName()` should return a pointer to an `mpe_SocketHostEntry` structure. The caller must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated, so the caller should copy any information it needs before calling any other socket function. Otherwise, `mpeos_socketGetHostByName()` should return a NULL pointer and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

`MPE_SOCKET_EHOSTNOTFOUND`

indicates the host is unknown.

`MPE_ENODATA`

specifies the server recognized the request and the `name`, but no address is available. Another type of request to the name server for the domain might return an answer.

`MPE_SOCKET_ENORECOVERY`

indicates an unexpected server failure occurred and cannot be recovered.

`MPE_SOCKET_ETRYAGAIN`

indicates a temporary and possibly transient error occurred, such as a failure of a server to respond.

description `mpeos_socketGetHostByName()` should get an entry containing addresses of address family `MPE_SOCKET_AF_INET4` or `MPE_SOCKET_AF_INET6` for the host specified by `name`. This information is considered to be stored in a database that can be accessed sequentially or randomly. Implementation of this database is unspecified.

`name` should be a node name. The behavior of `mpeos_socketGetHostByName()` when passed a numeric address string is unspecified. For IPv4, a numeric address string should be in dotted-decimal notation.

If `name` is not a numeric address string and is an alias for a valid host name, then `mpeos_socketGetHostByName()` should return information about the host name to which the alias refers, and the name should be included in the list of aliases returned.

related function(s) `mpeos_socketGetHostByAddr`
`mpeos_socketGetHostName`
`mpeos_socketGetLastError`

mpeos_socketGetHostName

gets the standard host name for the current machine.

syntax `int mpeos_socketGetHostName(
 char * name,
 size_t namelen);`

parameter(s) `name` is an output pointer to the buffer into which the host name is written (returned).

`namelen` specifies the size in characters of the buffer pointed to by `name`.

value returned If the call is successful, `mpeos_socketGetHostName()` should return 0. Otherwise, `mpeos_socketGetHostName()` should return -1.

description `mpeos_socketGetHostName()` should get the standard host name for the current machine. `namelen` should specify the size of the array pointed to by `name`. If `namelen` is not large enough to hold the host name, the returned name should be NULL-terminated or truncated.

The returned name is null-terminated and truncated if insufficient space is provided.

Host names are limited to MPE_SOCKET_MAXHOSTNAMELEN bytes.

related function(s) `mpeos_socketGetHostByAddr`
`mpeos_socketGetHostByName`
`mpeos_socketGetLastError`

mpeos_socketGetLastError

gets the error status of the last socket function.

syntax int mpeos_socketGetLastError(void);

parameter(s) none

value returned mpeos_socketGetLastError() should return the error status of the last socket function that failed.

description mpeos_socketGetLastError() should get the error status of the last socket function that failed.

related function(s)

- mpeos_socketAccept
- mpeos_socketBind
- mpeos_socketClose
- mpeos_socketConnect
- mpeos_socketCreate
- mpeos_socketGetHostByAddr
- mpeos_socketGetHostByName
- mpeos_socketGetOpt
- mpeos_socketGetPeerName
- mpeos_socketGetSockName
- mpeos_socketIoctl
- mpeos_socketListen
- mpeos_socketNtoP
- mpeos_socketPtoN
- mpeos_socketRecv
- mpeos_socketRecvFrom
- mpeos_socketSelect
- mpeos_socketSend
- mpeos_socketSendTo
- mpeos_socketSetOpt
- mpeos_socketShutdown

mpeos_socketGetOpt

manipulates options associated with a socket.

syntax int mpeos_socketGetOpt(
 mpe_Socket *socket*,
 int *level*,
 int *option_name*,
 void * *option_value*,
 mpe_SocketSockLen * *option_len*);

| | | |
|---------------------|--------------------------------|--|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. mpe_Socket is described in the <i>mpe_Socket</i> section. |
| | <i>level</i> | specifies the protocol level at which the option resides. |
| | <i>option_name</i> | specifies a single option to be retrieved. Currently, the following options are supported for <i>option_name</i> : |
| | MPE_SOCKET_IPV4_MULTICAST_IF | specifies the outgoing interface (mpe_SocketIPv4Addr). |
| | MPE_SOCKET_IPV4_MULTICAST_LOOP | enables or disables the local loopback of multicast datagrams (unsigned char). |
| | MPE_SOCKET_IPV4_MULTICAST_TTL | sets or reads the time-to-live value of outgoing multicast datagrams (unsigned char). |
| | MPE_SOCKET_IPV6_MULTICAST_HOPS | sets or reads the hop limit of outgoing multicast datagrams (unsigned int). |
| | MPE_SOCKET_IPV6_MULTICAST_IF | specifies the outgoing interface (mpe_SocketIPv6Addr). |
| | MPE_SOCKET_IPV6_MULTICAST_LOOP | enables or disables the local loopback of multicast datagrams (unsigned int). |
| | MPE_SOCKET_SO_BROADCAST | permits the sending of broadcast datagrams (int). |
| | MPE_SOCKET_SO_DEBUG | enables debug tracing (int). |
| | MPE_SOCKET_SO_DONTROUTE | bypasses routing table lookup (int). |
| | MPE_SOCKET_SO_ERROR | gets the pending error and clears the error (int). |
| | MPE_SOCKET_SO_KEEPALIVE | periodically tests if the connection is still alive (int). |

| | |
|---------------------|--|
| | MPE_SOCKET_SO_LINGER lingers on close if there is data to send (<i>mpe_SocketLinger</i>). |
| | MPE_SOCKET_SO_OOBINLINE leaves the received out-of-band data inline (int). |
| | MPE_SOCKET_SO_RCVBUF controls the size of the receive buffer (int). |
| | MPE_SOCKET_SO_RCVLOWAT controls the minimum number of bytes to process for socket input operations (int). |
| | MPE_SOCKET_SO_RCVTIMEO controls the time-out value for input operations. (<i>mpe_TimeVal</i>). |
| | MPE_SOCKET_SO_REUSEADDR allows a local address reuse (int). |
| | MPE_SOCKET_SO_SNDBUF controls the send buffer size (int). |
| | MPE_SOCKET_SO SNDLOWAT controls the minimum number of bytes to process for socket output operations (int). |
| | MPE_SOCKET_SO SNDTIMEO controls the time-out value specifying the amount of time an output function blocks (<i>mpe_TimeVal</i>). |
| | MPE_SOCKET_SO_TYPE reports the socket type (int). |
| | MPE_SOCKET_TCP_NODELAY disables the Nagle algorithm (int). |
| <i>option_value</i> | is an output pointer to storage sufficient to hold the value of the option. For Boolean options, a zero value indicates the option is disabled and a non-zero value indicates the option is enabled. |
| <i>option_len</i> | is an input and output pointer to the length of the buffer pointed to by <i>option_value</i> . When the call returns, <i>mpeos_socketGetOpt()</i> updates <i>option_len</i> to indicate the number of bytes actually copied to <i>option_value</i> . <i>mpe_SocketSockLen</i> is described in the <i>mpe_SocketSockLen</i> section. |

value returned If the call is successful, `mpeos_socketGetOpt()` should return 0. Otherwise, `mpeos_socketGetOpt()` should return -1 and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

`MPE_SOCKET_EBADF`

indicates an invalid file descriptor.

`MPE_EINVAL`

indicates at least one input parameter to the function has an invalid value.

`MPE_SOCKET_ENOBUFS`

indicates the function failed due to insufficient available resources in the system.

`MPE_SOCKET_ENOPROTOOPT`

indicates the option is not supported by the protocol.

`MPE_SOCKET_ENOTSOCK`

indicates that a parameter does not refer to a socket.

description

`mpeos_socketGetOpt()` should manipulate the options associated with a socket.

`mpeos_socketGetOpt()` should retrieve the value for the option specified by `option_name` for the socket specified by `socket`. If the size of the option value is greater than `option_len`, `mpeos_socketGetOpt()` should silently truncate the value stored in the object pointed to by `option_value`. Otherwise, the object pointed to by `option_len` should be modified to indicate the actual length of the value.

`level` specifies the protocol level at which the option resides. To retrieve options at the socket level, specify `level` as `MPE_SOCKET_SOL_SOCKET`. To retrieve options at other levels, supply the appropriate level identifier for the protocol controlling the option. For example, to indicate an option is interpreted by the Transmission Control Protocol (TCP), set `level` to `MPE_SOCKET IPPROTO_TCP`.

related function(s)

`mpeos_socketGetLastError`

`mpeos_socketSetOpt`

mpeos_socketGetPeerName

gets the peer address of the specified socket.

syntax `int mpeos_socketGetPeerName(
 mpe_Socket socket,
 mpe_SocketSockAddr * address,
 mpe_SocketSockLen * address_len);`

parameter(s) `socket` specifies the file descriptor associated with the socket in which to get the peer address. `mpe_Socket` is described in the *mpe_Socket* section.

`address` is an output pointer to an `mpe_SocketSockAddr` structure containing the peer address. `mpe_SocketSockAddr` is described in the *mpe_SocketSockAddr* section.

`address_len` is an input or output pointer to the length of the `mpe_SocketSockAddr` structure pointed to by `address`. `mpe_SocketSockLen` is described in the *mpe_SocketSockLen* section.

value returned If the call is successful, `mpeos_socketGetPeerName()` should return 0. Otherwise, `mpeos_socketGetPeerName()` should return -1 and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

`MPE_SOCKET_EBADF` indicates an invalid file descriptor.

`MPE_EINVAL` specifies either the `socket` has been shut down, bound to an address, or the protocol does not support binding to a new address.

`MPE_SOCKET_ENOBUFS` indicates the function failed due to insufficient available resources in the system.

`MPE_SOCKET_ENOTSOCK` indicates that a parameter does not refer to a socket.

description `mpeos_socketGetPeerName()` should do these procedures in the following order:

1. Get the peer address of the specified socket.
2. Store the peer address in the `mpe_SocketSockAddr` structure pointed to by `address`.
3. Store the length of this address in the object pointed to by `address_len`.

If the actual length of the address is greater than the length of the supplied `mpe_SocketSockAddr` structure, `mpeos_socketGetPeerName()` should truncate the stored address.

If the protocol permits connections by unbound clients and the peer is not bound, then the value stored in the object pointed to by `address` is unspecified.

related function(s) `mpeos_socketGetLastError`

mpeos_socketGetSockName

gets the locally-bound name of the specified socket.

syntax `int mpeos_socketGetSockName(
 mpe_Socket socket,
 mpe_SocketSockAddr * address,
 mpe_SocketSockLen * address_len);`

| | | |
|---------------------|--------------------|---|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket in which to get the locally-bound name. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>address</i> | is an output pointer to an <i>mpe_SocketSockAddr</i> structure where the name should be returned. <i>mpe_SocketSockAddr</i> is described in the <i>mpe_SocketSockAddr</i> section. |
| | <i>address_len</i> | is an input or output pointer to the length of the <i>mpe_SocketSockAddr</i> structure pointed to by <i>address</i> . When the call returns, <i>mpeos_socketGetSockName()</i> should update <i>address_len</i> to reflect the length of the name actually copied to <i>address</i> . <i>mpe_SocketSockLen</i> is described in the <i>mpe_SocketSockLen</i> section. |

value returned If the call is successful, *mpeos_socketGetSockName()* should get 0, *address* should point to the address of the socket, and *address_len* should point to the length of the address. Otherwise, *mpeos_socketGetSockName()* should return -1 and by making a call to *mpeos_socketGetLastError()* one of the following error codes can be retrieved:

| | |
|----------------------------------|--|
| <code>MPE_SOCKET_EBADF</code> | indicates an invalid file descriptor. |
| <code>MPE_EINVAL</code> | specifies either the <i>socket</i> has been shut down, bound to an address, or the protocol does not support binding to a new address. |
| <code>MPE_SOCKET_ENOBUFS</code> | indicates the function failed due to insufficient available resources in the system. |
| <code>MPE_SOCKET_ENOTSOCK</code> | indicates that a parameter does not refer to a socket. |

description *mpeos_socketGetSockName()* should do these procedures in the following order:

1. Get the locally-bound name of the specified socket.
2. Store this address in the *mpe_SocketSockAddr* structure pointed to by *address*.
3. Store the length of this address in the object pointed to by *address_len*.

If the actual length of the address is greater than the length of the supplied `mpe_SocketSockAddr` structure, `mpeos_socketGetSockName()` should truncate the stored address.

If the socket has not been bound to a local name, the value stored in the object pointed to by `address` is unspecified.

related function(s) `mpeos_socketConnect`

mpeos_socketHtoNL

converts a 32-bit quantity from host byte order to network byte order.

syntax `uint32_t mpeos_socketHtoNL(uint32_t hostlong);`

parameter(s) `hostlong` specifies the 32-bit value in host byte order.

value returned `mpeos_socketHtoNL()` should return `hostshort` converted to network byte order.

description `mpeos_socketHtoNL()` should convert a 32-bit quantity from host byte order to network byte order.

related function(s) `mpeos_socketHtoNS`
`mpeos_socketNtoHL`
`mpeos_socketNtoHS`

mpeos_socketHtoNS

converts a 16-bit quantity from host byte order to network byte order.

syntax `uint16_t mpeos_socketHtoNS(uint16_t hostshort);`

parameter(s) `hostshort` specifies the 16-bit value in host byte order.

value returned `mpeos_socketHtoNS()` should return `hostshort` converted to network byte order.

description `mpeos_socketHtoNS()` should convert a 16-bit quantity from host byte order to network byte order.

related function(s)
`mpeos_socketHtoNL`
`mpeos_socketNtoHL`
`mpeos_socketNtoHS`

mpeos_socketInit

initializes the networking API.

syntax mpe_Bool mpeos_socketInit(void);

parameter(s) none

value returned If the networking Application Programming Interface (API) is successfully initialized, `mpeos_socketInit()` should return TRUE. If the socket subsystem could not be initialized, `mpeos_socketInit()` should return FALSE.

description `mpeos_socketInit()` should initialize the networking API. `mpeos_socketInit()` must be called before calling any other functions with a prefix of `mpeos_socket`.

related function(s) `mpeos_socketAccept`
`mpeos_socketBind`
`mpeos_socketConnect`

mpeos_socketIoctl

performs a variety of control functions on a socket.

syntax `int mpeos_socketIoctl(
 mpe_Socket socket,
 int request,
 ...);`

| | | |
|----------------------------|----------------------------------|--|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>request</i> | specifies the request for the control function to perform. Currently, the following values are supported for <i>request</i> : |
| | <code>MPE_SOCKET_FIONBIO</code> | specifies the nonblocking flag for the socket is cleared or turned on, depending on whether the third argument points to a zero or non-zero integer value, respectively. |
| | <code>MPE_SOCKET_FIONREAD</code> | returns in the integer pointed to by the third argument, the number of bytes currently in the socket receive buffer. |
| | <i>arg</i> | specifies additional information needed to perform the requested function. The type of <i>arg</i> depends on the particular control request. |
| value returned | | If the call is successful, <code>mpeos_socketIoctl()</code> should return 0. Otherwise, <code>mpeos_socketIoctl()</code> should return -1 and by making a call to <code>mpeos_socketGetLastError()</code> one of the following error codes can be retrieved: |
| | <code>MPE_SOCKET_EBADF</code> | indicates an invalid file descriptor. |
| | <code>MPE EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| | <code>MPE_SOCKET_ENOTSOCK</code> | indicates that a parameter does not refer to a socket. |
| description | | <code>mpeos_socketIoctl()</code> should perform a variety of control functions on a socket. |
| related function(s) | | <code>mpeos_socketGetOpt</code> <code>mpeos_socketSetOpt</code> |

mpeos_socketListen

marks a socket as accepting connections.

syntax int mpeos_socketListen(
 mpe_Socket *socket*,
 int *backlog*);

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. mpe_Socket is described in the <i>mpe_Socket</i> section. |
| | <i>backlog</i> | specifies the requested maximum number of connections to allow in the socket's listen queue. |

value returned If the call is successful, `mpeos_socketListen()` should return 0. Otherwise, `mpeos_socketListen()` should return -1 and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

| | |
|---------------------|--|
| MPE_SOCKET_EBADF | indicates an invalid file descriptor. |
| MPE_EINVAL | specifies either the <i>socket</i> has been shut down, bound to an address, or the protocol does not support binding to a new address. |
| MPE_SOCKET_ENOBUFS | indicates the function failed due to insufficient available resources in the system. |
| MPE_SOCKET_ENOTSOCK | indicates that a parameter does not refer to a socket. |

description `mpeos_socketListen()` should mark a connection-mode socket, specified by *socket*, as accepting connections.

backlog provides a hint to the implementation, which should be used to limit the number of outstanding connections in the socket's listen queue. Implementations may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog* value should result in a larger or equal length of the listen queue.

The implementation may include incomplete connections in the listen queue. The limits on the number of incomplete connections and completed connections queued may be different.

The implementation may have an upper limit on the length of the listen queue-either global or per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

If `mpeos_socketListen()` is called with a *backlog* argument value of less than 0, `mpeos_socketListen()` should behave as if called with a *backlog* value of 0.

A *backlog* of 0 may allow the socket to accept connections, in which case the length of the listen queue may be set to an implementation-defined minimum value.

related function(s) [mpeos_socketCreate](#)
[mpeos_socketBind](#)

mpeos_socketNtoA

converts an Internet address to an ASCII string.

syntax `char * mpeos_socketNtoA(mpe_SocketIPv4Addr inaddr);`

parameter(s) `inaddr` specifies the Internet host address to convert.
`mpe_SocketIPv4Addr` is described in the
`mpe_SocketIPv4Addr` section.

value returned `mpeos_socketNtoA()` should return a pointer to the Internet address in Internet standard dot notation.

description `mpeos_socketNtoA()` should convert the Internet address stored in `inaddr` into an American National Standard Code for Information Interchange (ASCII) string representing the address in dot notation (for example, 127.0.0.1).

related function(s) `mpeos_socketAtoN`

mpeos_socketNtoHL

converts a 32-bit quantity from network byte order to host byte order.

syntax `uint32_t mpeos_socketNtoHL(uint32_t netlong);`

parameter(s) `netlong` specifies the 32-bit value in network byte order.

value returned `mpeos_socketNtoHL()` should return the value of `netlong` converted to host byte order.

description `mpeos_socketNtoHL()` should convert a 32-bit quantity from network byte order to host byte order.

related function(s) `mpeos_socketHtoNL`
`mpeos_socketHtoNS`
`mpeos_socketNtoHS`

mpeos_socketNtoHS

converts a 16-bit quantity from network byte order to host byte order.

syntax `uint16_t mpeos_socketNtoHS(uint16_t netshort);`

parameter(s) `netshort` specifies the 16-bit value in network byte order.

value returned `mpeos_socketNtoHS()` should return the value of `netshort` converted to host byte order.

description `mpeos_socketNtoHS()` should convert a 16-bit quantity from network byte order to host byte order.

related function(s) `mpeos_socketHtoNL`
`mpeos_socketHtoNS`
`mpeos_socketNtoHL`

mpeos_socketNtoP

converts a numeric address into a text string.

syntax `const char * mpeos_socketNtoP(int af, const void * src, char * dst, size_t size);`

| | | |
|---------------------|-------------|--|
| parameter(s) | <i>af</i> | specifies the address family. Currently, the values defined for <i>af</i> are either MPE_SOCKET_AF_INET4 or MPE_SOCKET_AF_INET6. |
| | <i>src</i> | is an input pointer to the source form of the address. |
| | <i>dst</i> | is an output pointer to the destination form of the address. |
| | <i>size</i> | specifies the size of the buffer pointed to by <i>dst</i> . |

value returned If the call is successful, `mpeos_socketNtoP()` should return a pointer to the buffer containing the text string. Otherwise, `mpeos_socketNtoP()` should return NULL and by making a call to `mpeos_socketGetLastError()` the following error code can be retrieved:

`MPE_SOCKET_ENOSPC`
indicates the size of the result buffer is inadequate.

description `mpeos_socketNtoP()` should convert a numeric address into a text string suitable for presentation.

src points to a buffer holding either an IPv4 address (if *af* is MPE_SOCKET_AF_INET4) or an IPv6 address (if *af* is MPE_SOCKET_AF_INET6).

dst points to a buffer where the function stores the resulting text string and should not be NULL.

size specifies the size of this buffer, which should be large enough to hold the text string (MPE_SOCKET_INET4_ADDRSTRLEN characters for IPv4, MPE_SOCKET_INET6_ADDRSTRLEN characters for IPv6).

related function(s) `mpeos_socketPtoN`

mpeos_socketPtoN

converts a text string to a numeric address.

syntax `int mpeos_socketPtoN(
 int af,
 const char * src,
 void * dst);`

| | | |
|---------------------|------------|--|
| parameter(s) | <i>af</i> | specifies the address family. Currently, the values defined for <i>af</i> are either MPE_SOCKET_AF_INET4 or MPE_SOCKET_AF_INET6. |
| | <i>src</i> | is an input pointer to the source form of the address. |
| | <i>dst</i> | is an output pointer to the destination form of the address |

value returned If the conversion is successful, `mpeos_socketPtoN()` should return 1, with the address pointed to by *dst* in network byte order. If the input is not a valid IPv4 dotted-decimal or IPv6 address string, `mpeos_socketPtoN()` should return 0. If an error occurs, it should return -1 and by making a call to `mpeos_socketGetLastError()`.

description `mpeos_socketPtoN()` should convert an address in its standard text presentation form into the numeric binary form.

src points to the string being passed in. *dst* points to a buffer into which the function stores the numeric address; this should be large enough to hold the numeric address (32 bits for MPE_SOCKET_AF_INET4, 128 bits for MPE_SOCKET_AF_INET6).

If *af* is MPE_SOCKET_AF_INET4, the *src* string should be in the standard IPv4 dotted-decimal form; for example, ddd.ddd.ddd.ddd, where ddd is a one to three digit decimal number between 0 and 255 inclusive.

`mpeos_socketPtoN()` does not accept other formats (such as the octal numbers, hexadecimal numbers, or fewer than four numbers).

If *af* is MPE_SOCKET_AF_INET6, the *src* string should be in one of the following standard IPv6 text forms:

- ◆ The preferred form is *x:x:x:x:x:x:x*, where *x* is the hexadecimal values of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted, but there should be at least one numeral in every field.
- ◆ A string of contiguous zero fields in the preferred form can be shown as *::*. The *::* can appear only once in an address. Unspecified addresses (*0:0:0:0:0:0:0:0*) may be represented simply as *::*.
- ◆ A third form is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes where the *x* is the hexadecimal values of the six high-order 16-bit pieces of the address, and the *d* is the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

A more extensive description of the standard representations of IPv6 addresses can be found in the Network Working Group Request for Comments 2373.

related function(s) mpeos_socketNtoP

mpeos_socketRecv

receives a message from a socket.

syntax `size_t mpeos_socketRecv(
 mpe_Socket socket,
 void * buffer,
 size_t length,
 int flags);`

| | | |
|---------------------|---------------------|---|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>buffer</i> | is an output pointer to a buffer where the message should be stored. |
| | <i>length</i> | specifies the length in bytes of the buffer pointed to by <i>buffer</i> . |
| | <i>flags</i> | specifies the type of message reception. Values of <i>flags</i> are formed by logically OR'ing zero or more values. Currently, the following values are defined for flag: |
| | MPE_SOCKET_MSG_PEEK | peeks at an incoming message. The data is treated as unread and the next <i>mpeos_socketRecv()</i> or similar function should still return this data. |
| | MPE_SOCKET_MSG_OOB | sends or receives out-of-band data on sockets that support out-of-band data. |

value returned If the call is successful, *mpeos_socketRecv()* should return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *mpeos_socketRecv()* should return 0. Otherwise, *mpeos_socketRecv()* should return -1 and by making a call to *mpeos_socketGetLastError()* one of the following error codes can be retrieved:

| | |
|-----------------------|--|
| MPE_SOCKET_EBADF | indicates an invalid file descriptor. |
| MPE_SOCKET_ECONNRESET | indicates the remote host reset the connection request. |
| MPE_EINVAL | indicates at least one input parameter to the function has an invalid value. |
| MPE_SOCKET_ENOBUFS | indicates the function failed due to insufficient available resources in the system. |
| MPE_ENOMEM | indicates the function failed due to insufficient memory resources. |

MPE_SOCKET_ENOTSOCK

indicates that a parameter does not refer to a socket.

MPE_ETHREADDEATH

indicates the thread executing the function has been marked for death.

description

`mpeos_socketRecv()` should receive a message from a connection-mode or connectionless-mode socket. `mpeos_socketRecv()` is normally used with connected sockets because `mpeos_socketRecv()` does not permit the application to retrieve the source address of received data.

`mpeos_socketRecv()` should return the length of the message written to the buffer specified by *buffer*. For message-based sockets, such as `MPE_SOCKET_DATAGRAM`, the entire message should be read in a single operation. If a message is too long to fit in the supplied buffer, and `MPE_SOCKET_MSG_PEEK` is not set in *flags*, the excess bytes should be discarded. For stream-based sockets, such as `MPE_SOCKET_STREAM`, message boundaries should be ignored. In this case, data should be returned to the user as soon as it becomes available, and no data should be discarded.

If no messages are available at the socket, `mpeos_socketRecv()` should block until a message arrives.

related function(s)

`mpeos_socketConnect`
`mpeos_socketRecvFrom`

mpeos_socketRecvFrom

receives a message and retrieves the source address from a socket.

```
syntax size_t mpeos_socketRecvFrom(
    mpe_Socket socket,
    void * buffer,
    size_t length,
    int flags,
    mpe_SocketSockAddr * address,
    mpe_SocketSockLen * address_len );
```

| | | |
|---------------------|---------------------|---|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>buffer</i> | is an output pointer to the buffer where the message should be stored. |
| | <i>length</i> | specifies the length in bytes of the buffer pointed to by <i>buffer</i> . |
| | <i>flags</i> | specifies the type of message reception. Values of <i>flags</i> are formed by logically OR'ing zero or more values. Currently, the following values are defined for flag: |
| | MPE_SOCKET_MSG_PEEK | peeks at an incoming message. The data is treated as unread and the next <i>mpeos_socketRecvFrom()</i> or similar function should still return this data. |
| | MPE_SOCKET_MSG_OOB | sends or receives out-of-band data on sockets that support out-of-band data. |
| | <i>address</i> | is an output pointer to a <i>mpe_SocketSockAddr</i> structure in which the sending address is to be stored. If <i>address</i> is NULL, no address is returned. The length and format of the address depend on the address family of the socket. <i>mpe_SocketSockAddr</i> is described in the <i>mpe_SocketSockAddr</i> section. |
| | <i>address_len</i> | is an input or output pointer to the length of the <i>mpe_SocketSockAddr</i> structure pointed to by <i>address</i> . <i>mpe_SocketSockLen</i> is described in the <i>mpe_SocketSockLen</i> section. |

value returned If the call is successful, *mpeos_socketRecvFrom()* should return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *mpeos_socketRecvFrom()* should return 0. Otherwise, *mpeos_socketRecvFrom()* should return -1 and by making a call to *mpeos_socketGetLastError()* one of the following error codes can be retrieved:

MPE_SOCKET_EBADF
indicates an invalid file descriptor.

| | |
|---------------------|--|
| MPE_EINVAL | indicates at least one input parameter to the function has an invalid value. |
| MPE_SOCKET_ENOBUFS | indicates the function failed due to insufficient available resources in the system. |
| MPE_ENOMEM | indicates the function failed due to insufficient memory resources. |
| MPE_SOCKET_ENOTSOCK | indicates that a parameter does not refer to a socket. |
| MPE_ETHREADDEATH | indicates the thread executing the function has been marked for death. |

description

mpeos_socketRecvFrom() should receive a message from a connection-mode or connectionless-mode socket. mpeos_socketRecvFrom() is normally used with connectionless-mode sockets to accept application retrieval of the source address of received data. mpeos_socketRecvFrom() should return the length of the message written to the buffer pointed to by *buffer*. For message-based sockets, such as MPE_SOCKET_DATAGRAM, the entire message should be read in a single operation. If a message is too long to fit in the supplied buffer, and MPE_SOCKET_MSG_PEEK is not set in *flags*, mpeos_socketRecvFrom() should discard the excess bytes. For stream-based sockets, such as MPE_SOCKET_STREAM, mpeos_socketRecvFrom() should ignore message boundaries. In this case, data should be returned to the user as soon as it becomes available, and no data should be discarded.

Not all protocols provide the source address for messages. If *address* is not a NULL pointer and the protocol provides the source address of messages, the source address of the received message should be stored in the mpe_SocketSockAddr structure pointed to by *address*, and the length of this address should be stored in the object pointed to by *address_len*.

If the actual length of the address is greater than the length of the supplied mpe_SocketSockAddr structure, mpeos_socketRecvFrom() should truncate the stored address.

If *address* is not a NULL pointer and the protocol does not provide the source address of messages, the value stored in the object pointed to by *address* is unspecified.

If no messages are available at the socket, mpeos_socketRecvFrom() should block until a message arrives.

related function(s)

mpeos_socketConnect
mpeos_socketRecv

mpeos_socketSelect

examines file descriptor sets.

```
syntax int mpeos_socketSelect(
    int numfds,
    mpe_SocketFDSet * readfds,
    mpe_SocketFDSet * writefds,
    mpe_SocketFDSet * errorfds,
    const mpe_TimeVal * timeout );
```

| | | |
|---------------------|-----------------|---|
| parameter(s) | <i>numfds</i> | specifies the range of descriptors to be tested. The first <i>numfds</i> descriptors should be checked in each set; that is, the descriptors from zero through <i>numfds</i> -1 in the descriptor sets should be examined. |
| | <i>readfds</i> | is an input pointer to an object of type <i>mpe_SocketFDSet</i> that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read. <i>readfds</i> may also be a NULL pointer. <i>mpe_SocketFDSet</i> is described in the <i>mpe_SocketFDSet</i> section. |
| | <i>writefds</i> | is an input or output pointer to an object of type <i>mpe_SocketFDSet</i> that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write. <i>writefds</i> may also be a NULL pointer. <i>mpe_SocketFDSet</i> is described in the <i>mpe_SocketFDSet</i> section. |
| | <i>errorfds</i> | is an input or output pointer to an object of type <i>mpe_SocketFDSet</i> that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending. <i>errorfds</i> may also be a NULL pointer. <i>mpe_SocketFDSet</i> is described in the <i>mpe_SocketFDSet</i> section. |
| | <i>timeout</i> | is an input pointer to the time-out period in seconds and microseconds. <i>mpe_TimeVal</i> is described in the <i>mpe_TimeVal</i> section. |

value returned If the call is successful, *mpeos_socketSelect()* should return the total number of bits set in the bit mask. Otherwise, *mpeos_socketSelect()* should return -1 and by making a call to *mpeos_socketGetLastError()* one of the following error codes can be retrieved:

- MPE_SOCKET_EBADF indicates an invalid file descriptor.
- MPE_EINVAL indicates either at least one input parameter to the function has an invalid value or *numfds* is less than 0 or greater than MPE_SOCKET_FD_SETSIZE.

MPE_ETHREADDEATH

indicates the thread executing the function has been marked for death.

description

mpeos_socketSelect() should examine the file descriptor sets whose addresses are passed in *readfds*, *writefd*s, and *errfd*s to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The behavior of mpeos_socketSelect() on non-socket file descriptors is unspecified.

When the call completes successfully, mpeos_socketSelect() should modify the objects pointed to by *readfds*, *writefd*s, and *errfd*s to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively, and should return the total number of ready descriptors in all the output sets. For each file descriptor less than *numfd*s, the corresponding bit should be set on successful completion if the bit was set on input and the associated condition is true for that file descriptor.

If none of the selected descriptors are ready for the requested operation, mpeos_socketSelect() should block until one of the following occurs:

- ◆ at least one of the requested operations becomes ready
- ◆ the time-out occurs
- ◆ a call is interrupted by a signal

timeout controls how long mpeos_socketSelect() should take before timing out. If *timeout* is not a NULL pointer, mpeos_socketSelect() specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, mpeos_socketSelect() should return. If *timeout* is a NULL pointer, then the call to mpeos_socketSelect() should block indefinitely until at least one descriptor meets the specified criteria. To effect a poll, *timeout* should not be a NULL pointer, and should point to a zero-valued mpe_timeVal structure.

Implementations may place limitations on the maximum time-out interval supported. If *timeout* specifies a time-out interval greater than the implementation-defined maximum value, the maximum value should be used as the actual time-out value. Implementations may also place limitations on the granularity of time-out intervals. If the requested time-out interval requires a finer granularity than the implementation supports, the actual time-out interval should be rounded up to the next supported value.

If *readfds*, *writefd*s, and *errfd*s are all NULL pointers and *timeout* is not a NULL pointer, mpeos_socketSelect() should block for the time specified, or until interrupted by a signal. If *readfds*, *writefd*s, and *errfd*s are all NULL pointers and *timeout* is a NULL pointer, mpeos_socketSelect() should block until interrupted by a signal.

If the call fails, the objects pointed to by *readfds*, *writefd*s, and *errorfd*s should not be modified. If the time-out interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by *readfds*, *writefd*s, and *errorfd*s should have all bits set to 0.

related function(s) mpeos_socketConnect

mpeos_socketSend

initiates sending a message from a socket to the socket's peer.

syntax

```
size_t mpeos_socketSend(
    mpe_Socket socket,
    const void * buffer,
    size_t length,
    int flags );
```

| | | |
|---------------------|---------------|---|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. mpe_Socket is described in the <i>mpe_Socket</i> section. |
| | <i>buffer</i> | is an input pointer to the buffer containing the message to send. |
| | <i>length</i> | specifies the length of the message in bytes. |
| | <i>flags</i> | specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more values. Currently, the following value is defined for flag: |
| | | MPE_SOCKET_MSG_OOB sends or receives out-of-band data on sockets that support out-of-band data. |

value returned If the call is successful, `mpeos_socketSend()` should return the number of bytes sent. Otherwise, `mpeos_socketSend()` should return -1 and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

| | |
|------------------------|--|
| MPE_SOCKET_EBADF | indicates an invalid file descriptor. |
| MPE_SOCKET_ENETDOWN | indicates the local network interface used to reach the destination is down. |
| MPE_SOCKET_ENETUNREACH | indicates no route to the network is present. |
| MPE_SOCKET_ENOBUFS | indicates the function failed due to insufficient available resources in the system. |
| MPE_SOCKET_ENOTSOCK | indicates that a parameter does not refer to a socket. |
| MPE_ETHREADDEATH | indicates the thread executing the function has been marked for death. |

description `mpeos_socketSend()` should initiate transmission of a message from the specified socket to the socket's peer. `mpeos_socketSend` should send a message only when the socket is connected (including when the peer of a connectionless socket has been set via `mpeos_socketConnect()`).

The length of the message to be sent is specified by *length*. If the message is too long to pass through the underlying protocol, `mpeos_socketSend()` should fail and no data should be transmitted.

Successful completion of a call to `mpeos_socketSend()` does not guarantee delivery of the message. A return value of -1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted, `mpeos_socketSend()` should block until space is available. The `mpeos_socketSelect()` function can be used to determine when it is possible to send more data.

related function(s) `mpeos_socketConnect`

mpeos_socketSendTo

sends a message through a connection-mode or connectionless-mode socket.

syntax `size_t mpeos_socketSendTo(mpe_Socket socket, const void * message, size_t length, int flags, const mpe_SocketSockAddr * dest_addr, mpe_SocketSockLen dest_len);`

| | | |
|---------------------|------------------|--|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>message</i> | is an input pointer to a buffer containing the message to be sent. |
| | <i>length</i> | specifies the size of the message in bytes. |
| | <i>flags</i> | specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more values. Currently, the following value is defined for flag: <code>MPE_SOCKET_MSG_OOB</code> sends or receives out-of-band data on sockets that support out-of-band data. |
| | <i>dest_addr</i> | is an input pointer to a <i>mpe_SocketSockAddr</i> structure containing the destination address. The length and format of the address depends on the address family of the socket. <i>mpe_SocketSockAddr</i> is described in the <i>mpe_SocketSockAddr</i> section. |
| | <i>dest_len</i> | specifies the length of the <i>mpe_SocketSockAddr</i> structure pointed to by <i>dest_addr</i> . <i>mpe_SocketSockLen</i> is described in the <i>mpe_SocketSockLen</i> section. |

value returned If the call is successful, `mpeos_socketSendTo()` should return the number of bytes sent. Otherwise, `mpeos_socketSendTo()` should return -1 and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

| | |
|-------------------------------------|--|
| <code>MPE_SOCKET_EBADF</code> | indicates an invalid file descriptor. |
| <code>MPE_EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| <code>MPE_SOCKET_ENETDOWN</code> | indicates the local network interface used to reach the destination is down. |
| <code>MPE_SOCKET_ENETUNREACH</code> | indicates no route to the network is present. |

MPE_SOCKET_ENOBUFS

indicates the function failed due to insufficient available resources in the system.

MPE_ENOMEM indicates the function failed due to insufficient memory resources.**MPE_SOCKET_ENOTSOCK**

indicates that a parameter does not refer to a socket.

MPE_ETHREADDEATH

indicates the thread executing the function has been marked for death.

description `mpeos_socketSendTo()` should send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message should be sent to the address specified by *dest_addr*. If the socket is connection-mode, *dest_addr* should be ignored.

When the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, `mpeos_socketSendTo()` should fail if the MPE_SOCKET_SO_BROADCAST option is not set for the socket.

Successful completion of a call to `mpeos_socketSendTo()` does not guarantee delivery of the message. A return value of -1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted, `mpeos_socketSendTo()` should block until space is available.

related function(s)

`mpeos_socketConnect`

`mpeos_socketSend`

mpeos_socketSetOpt

sets the specified option.

syntax int mpeos_socketSetOpt(
 mpe_Socket *socket*,
 int *level*,
 int *option_name*,
 const void * *option_value*,
 mpe_SocketSockLen *option_len*);

| | | |
|---------------------|---------------------------------|--|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. mpe_Socket is described in the <i>mpe_Socket</i> section. |
| | <i>level</i> | is the protocol level at which the option resides |
| | <i>option_name</i> | specifies a single option to be set. Currently, the following values are supported for <i>option_name</i> : |
| | MPE_SOCKET_IPV4_ADD_MEMBERSHIP | joins a multicast group (<i>mpe_SocketIPv4McastReq</i>). |
| | MPE_SOCKET_IPV4_DROP_MEMBERSHIP | leaves a multicast group (<i>mpe_SocketIPv4McastReq</i>). |
| | MPE_SOCKET_IPV4_MULTICAST_IF | specifies the outgoing interface (<i>mpe_SocketIPv4Addr</i>). |
| | MPE_SOCKET_IPV4_MULTICAST_LOOP | enables or disables the local loopback of multicast datagrams (unsigned char). |
| | MPE_SOCKET_IPV4_MULTICAST_TTL | sets or reads the time-to-live value of outgoing multicast datagrams (unsigned char). |
| | MPE_SOCKET_IPV6_ADD_MEMBERSHIP | joins a multicast group (<i>mpe_SocketIPv6McastReq</i>). |
| | MPE_SOCKET_IPV6_DROP_MEMBERSHIP | leaves a multicast group (<i>mpe_SocketIPv6McastReq</i>). |
| | MPE_SOCKET_IPV6_MULTICAST_HOPS | sets or reads the hop limit of outgoing multicast datagrams (unsigned int). |
| | MPE_SOCKET_IPV6_MULTICAST_IF | specifies the outgoing interface (<i>mpe_SocketIPv6Addr</i>). |
| | MPE_SOCKET_IPV6_MULTICAST_LOOP | enables or disables the local loopback of multicast datagrams (unsigned int). |
| | MPE_SOCKET_SO_BROADCAST | permits the sending of broadcast datagrams (int). |

| | |
|-------------------------|--|
| MPE_SOCKET_SO_DEBUG | enables debug tracing (int). |
| MPE_SOCKET_SO_DONTROUTE | bypasses routing table lookup (int). |
| MPE_SOCKET_SO_KEEPALIVE | periodically tests if the connection is still alive (int). |
| MPE_SOCKET_SO_LINGER | lingers on close if there is data to send (mpe_SocketLinger). |
| MPE_SOCKET_SO_OOBINLINE | leaves the received out-of-band data inline (int). |
| MPE_SOCKET_SO_RCVBUF | controls the size of the receive buffer (int). |
| MPE_SOCKET_SO_RCVLOWAT | controls the minimum number of bytes to process for socket input operations. (int). |
| MPE_SOCKET_SO_RCVTIMEO | controls the time-out value for input operations. (mpe_TimeVal). |
| MPE_SOCKET_SO_REUSEADDR | allows local address reuse (int). |
| MPE_SOCKET_SO_SNDBUF | controls the send buffer size (int). |
| MPE_SOCKET_SO SNDLOWAT | controls the minimum number of bytes to process for socket output operations. (int). |
| MPE_SOCKET_SO SNDTIMEO | controls the time-out value specifying the amount of time an output function blocks (mpe_TimeVal). |
| MPE_SOCKET_SO_TYPE | reports the socket type (int). |
| MPE_SOCKET_TCP_NODELAY | disables the Nagle algorithm (int). |
| <i>option_value</i> | is an input pointer to the new value for the option |
| <i>option_len</i> | specifies the length of option pointed to by <i>option_value</i> . mpe_SocketSockLen is described in the <i>mpe_SocketSockLen</i> section. |

value returned If the call is successful, `mpeos_socketSetOpt()` should return 0. Otherwise, `mpeos_socketSetOpt()` should return -1 and by making a call to `mpeos_socketGetLastError()` one of the following error codes can be retrieved:

| | |
|------------------------|--|
| MPE_SOCKET_EBADF | indicates an invalid file descriptor. |
| MPE_EINVAL | indicates at least one input parameter to the function has an invalid value. |
| MPE_SOCKET_ENOBUFS | indicates the function failed due to insufficient available resources in the system. |
| MPE_ENOMEM | indicates the function failed due to insufficient memory resources. |
| MPE_SOCKET_ENOPROTOOPT | indicates the option is not supported by the protocol. |
| MPE_SOCKET_ENOTSOCK | indicates that a parameter does not refer to a socket. |

description mpeos_socketSetOpt() should set the option specified by *option_name*, at the protocol level specified by *level*, to the value pointed to by *option_value* for the socket associated with the file descriptor specified by *socket*.

level specifies the protocol level at which the option resides. To set options at the socket level, specify *level* as MPE_SOCKET_SOL_SOCKET. To set options at other levels, supply the appropriate level identifier for the protocol controlling the option. For example, to indicate an option is interpreted by the Transmission Control Protocol (TCP), set *level* to MPE_SOCKET IPPROTO_TCP.

option_name specifies a single option to set. *option_name* and any specified options are passed uninterpreted to the appropriate protocol module for interpretations.

related function(s) mpeos_socketConnect
mpeos_socketSendTo

mpeos_socketShutdown

shuts down all or part of a full-duplex connection.

syntax `int mpeos_socketShutdown(
 mpe_Socket socket,
 int how);`

| | | |
|---------------------|---------------|---|
| parameter(s) | <i>socket</i> | specifies the file descriptor associated with the socket. <i>mpe_Socket</i> is described in the <i>mpe_Socket</i> section. |
| | <i>how</i> | specifies the type of shutdown. Currently, the following values are supported for <i>how</i> : |
| | | <code>MPE_SOCKET_SHUT_RD</code> disables further receive operations. |
| | | <code>MPE_SOCKET_SHUT_RDWR</code> disables further send and receive operations. |
| | | <code>MPE_SOCKET_SHUT_WR</code> disables further send operations. |

| | |
|-----------------------|--|
| value returned | If the call is successful, <code>mpeos_socketShutdown()</code> should return 0. Otherwise, <code>mpeos_socketShutdown()</code> should return -1 and by making a call to <code>mpeos_socketGetLastError()</code> one of the following error codes can be retrieved: |
| | <code>MPE_SOCKET_EBADF</code> indicates an invalid file descriptor. |
| | <code>MPE EINVAL</code> indicates at least one input parameter to the function has an invalid value. |
| | <code>MPE_SOCKET_ENOBUFS</code> indicates the function failed due to insufficient available resources in the system. |
| | <code>MPE_SOCKET_ENOTSOCK</code> indicates that a parameter does not refer to a socket. |

| | |
|--------------------|--|
| description | <code>mpeos_socketShutdown()</code> should cause all or part of a full-duplex connection on the socket associated with the file descriptor <i>socket</i> to be shut down. <code>mpeos_socketShutdown()</code> disables subsequent send and/or receive operations on a socket, depending on the value of <i>how</i> . |
|--------------------|--|

| | |
|----------------------------|---------------------------------|
| related function(s) | <code>mpeos_socketCreate</code> |
|----------------------------|---------------------------------|

mpeos_socketTerm

terminates the networking API.

syntax void mpeos_socketTerm(void);

parameter(s) none

value returned none

description mpeos_socketTerm() should terminate the network API. Use mpeos_socketTerm() when the networking API is no longer being used.

related function(s) mpeos_socketInit

POD Point-Of-Deployment (POD) API

Overview

The Point-Of-Deployment (POD) is a detachable device distributed by cable providers that connects to the home receiver. OpenCable refers to this device as a CableCard. The interface between the POD device and the receiver is specified by the OpenCable platform. POD functionality includes copy protection and signal demodulation.

The POD Application Programming Interface (API) provides a consistent interface to POD functionality regardless of the underlying operating system.

Before reading this chapter, you should be:

- ◆ familiar with the OCAP POD/Host interface and SCTE-28 POD/Host interface specifications

After reading this chapter, you should be able to port the POD functionality within the OCAP stack

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the POD functions:

| | |
|-------------|------------|
| MPE_EEVENT | MPE_EINVAL |
| MPE_ENODATA | MPE_ENOMEM |
| MPE_SUCCESS | |

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

MPE_EEVENT

`MPE_EEVENT` indicates the creating, deleting, or processing of events failed. `MPE_EEVENT` is defined as follows:

```
#define MPE_EEVENT OS_EEVENT
```

MPE_EINVAL

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENODATA

`MPE_ENODATA` indicates no data is available for the specified request. `MPE_ENODATA` is defined as follows:

```
#define MPE_ENODATA OS_ENODATA
```

MPE_ENOMEM

`MPE_ENOMEM` indicates the function failed due to insufficient memory resource. `MPE_ENOMEM` is defined as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_SUCCESS

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h`, `mpe_event.h`, `mpe_pod.h`, and `mpe_types.h`, are used by the POD functions:

| | |
|------------------------------|-----------------------------------|
| <code>mpe_Bool</code> | <code>mpe_Error</code> |
| <code>mpe_EventQueue</code> | <code>mpe_Mutex</code> |
| <code>mpe_PODAppInfo</code> | <code>mpe_PODDatabase</code> |
| <code>mpe_PODFeatures</code> | <code>mpe_PODFeatureParams</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The Multimedia Platform Environment (MPE) error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpe_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

`mpe_Mutex`

`mpe_Mutex` defines mutex-type binding. `mpe_Mutex` is defined in `mpeos_sync.h` as follows:

```
typedef os_Mutex mpe_Mutex;
```

`mpe_PODAppInfo`

`mpe_PODAppInfo` specifies data from an `application_info_cnf()` message. `mpe_PODAppInfo` is defined in `mpe_pod.h` as follows:

```
typedef struct {
    uint16_t ai_manufacturerId;
    uint16_t ai_versionNumber;
    uint8_t ai_appCount;
    uint8_t ai_apps[1];
} mpe_PODAppInfo;
```

where

`ai_manufacturerId`
specifies the POD manufacturer identifier.

`ai_versionNumber`
specifies the POD version number.

`ai_appCount` specifies the number of application to follow.

`ai_apps[1]` specifies 0 or more applications.

mpe_PODDatabase

`mpe_PODDatabase` specifies the main database container for all of the POD related information. `mpe_PODDatabase` is defined in `mpe_pod.h` as follows:

```
typedef struct {
    mpe_Bool pod_isReady;
    mpe_PODAppInfo *pod_appInfo;
    mpe_PODFeatures *pod_features;
    mpe_PODFeatureParams pod_featureParams;
} mpe_PODDatabase;
```

where

`pod_isReady` determines if the POD is present and initialized. If `pod_isReady` is TRUE, the POD is initialized. If `pod_isReady` is FALSE, the POD is not initialized. `mpe_Bool` is described in the *mpe_Bool* section.

`pod_appInfo` is a pointer to the POD application information. `mpe_PODAppInfo` is described in the *mpe_PODAppInfo* section.

`pod_features` is a pointer to the POD generic feature list. `mpe_PODFeatures` is described in the *mpe_PODFeatures* section.

`pod_featureParams` specifies the POD generic feature parameters. `mpe_PODFeatureParams` is described in the *mpe_PODFeatureParams* section.

mpe_PODFeatures

`mpe_PODFeatures` specifies data from a `feature_list()` response message. `mpe_PODFeatures` is defined in `mpe_pod.h` as follows:

```
typedef struct {
    uint8_t number;
    uint8_t featureIds[1];
} mpe_PODFeatures;
```

where

`number` specifies the number of generic features identifiers.

`featureIds[1]` specifies the feature identifiers of supported features.

mpe_PODFeatureParams

`mpe_PODFeatureParams` contains the current values of the generic features that are configurable between the CableCARD and the OpenCable Application Platform (OCAP) host device. It does not represent all the possible feature parameter messages defined in [ANSI/SCTE 28 2004 Host-POD Interface Standard](#), Section 8.12.5, but does include the parameters that are configurable by the CableCard and/or an OCAP application. `mpe_PODFeatureParams` is defined in `mpe_pod.h` as follows:

```
typedef struct {
    uint8_t rf_output_channel[2];
    uint8_t *pc_pin;
    uint8_t *pc_setting;
    uint8_t *ippv_pin;
    uint8_t time_zone_offset[2];
    uint8_t daylight_savings_ctrl;
    uint8_t ac_outlet_ctrl;
    uint8_t language_ctrl[3];
    uint8_t ratings_region;
    uint8_t reset_pin_ctrl;
    uint8_t cable_urls_length;
    uint8_t *cable_urls;
    uint8_t ea_location[3];
} mpe_PODFeatureParams;
```

where

| | |
|------------------------------------|--|
| <code>rf_output_channel[2]</code> | specifies the Radio Frequency (RF) output channel data and the user interface enable/disable flag. |
| <code>pc_pin</code> | is a pointer to the parental control pin. |
| <code>pc_setting</code> | is a pointer to the parental control channel settings. |
| <code>ippv_pin</code> | is a pointer to the Impulse-Pay-Per-View (IPPV) pin. |
| <code>time_zone_offset</code> | specifies the time-zone offset in minutes from Greenwich mean time. |
| <code>daylight_savings_ctrl</code> | specifies the daylight savings control. |
| <code>ac_outlet_ctrl</code> | specifies the Alternating Current (AC) outlet control. |
| <code>language_ctrl</code> | specifies the International Organization for Standardization (ISO) 639, 3-byte language code. |
| <code>ratings_region</code> | specifies the ratings region identifier. |
| <code>reset_pin_ctrl</code> | specifies the reset pin control indicator. |

`cable_urls_length`
specifies the total size of cable Uniform Resource Locator (URL) data.

`cable_urls` is a pointer to the cable URL.

`ea_location` specifies the Emergency Alert System (EAS) location code.

Events

POD events are used by the platform-specific layer to notifying the OCAP stack of asynchronous changes to the POD's generic features list or application information. The following POD events are used by the media functions:

MPE_POD_EVENT_GF_UPDATE

MPE_POD_EVENT_GF_UPDATE is sent to notify the host that a value has changed for a platform-specific feature. MPE_POD_EVENT_GF_UPDATE is defined as follows:

```
#define MPE_POD_EVENT_GF_UPDATE 1
```

MPE_POD_EVENT_APPINFO_UPDATE

MPE_POD_EVENT_APPINFO_UPDATE is sent to notify the host that a value has changed for a application feature. MPE_POD_EVENT_APPINFO_UPDATE is defined as follows:

```
#define MPE_POD_EVENT_APPINFO_UPDATE 2
```

Supported functions

The following POD functions need to be supported:

- `mpeos_podInit`
initiates communication between the POD and the host.
- `mpeos_podGetAppInfo`
identifies the pointer to the POD application information.
- `mpeos_podGetCCIBits`
gets the copy-control information for the service.
- `mpeos_podGetFeatures`
identifies the pointer to the POD generic feature list.
- `mpeos_podGetFeatureParam`
populates the internal POD database with the specified feature.
- `mpeos_podMMIClose`
closes a connection with the MMI resource on the POD.
- `mpeos_podMMIConnect`
establishes a connection with the MMI resource on the POD.
- `mpeos_podReceiveAPDU`
receives an APDU packet.
- `mpeos_podReceiveAPDU`
receives an APDU packet.
- `mpeos_podSASClose`
provides an optional unregister from a SAS application.
- `mpeos_podSASConnect`
establishes a connection between a host application and a POD application.
- `mpeos_podSendAPDU`
sends an APDU packet.
- `mpeos_podSetFeatureParam`
sets the feature in the host-POD interface.
- `mpeos_podUnregister`
unregisters for POD-specific asynchronous events.

mpeos_podInit

initiates communication between the POD and the host.

syntax `mpe_Error mpeos_podInit(mpe_PODDatabase * podDatabase);`

parameter(s) `podDatabase` is an input pointer to the MPE layer platform-independent POD information database used to cache the host-POD information. `mpe_PODDatabase` is described in the *mpe_PODDatabase* section.

value returned If the call is successful, `mpeos_podInit()` should return `MPE_SUCCESS`, along with a pointer to the POD database specified by `podDatabase`. Otherwise, it should return the following error code:

`MPE EINVAL` indicates `podDatabase` has an invalid value.

description `mpeos_podInit()` should perform any target-specific operations to enable interfacing with the POD device via the target host-POD devices stack interface. Depending on the platform implementation, this may include OCAP stack API call(s) to get the host-POD stack resources initialized, or it may simply involve OCAP stack API call(s) to access the data already exchanged between the host and POD during the platform initialization process.

related function(s) none

mpeos_podGetAppInfo

identifies the pointer to the POD application information.

syntax mpe_Error mpeos_podGetAppInfo(mpe_PODDatabase * podDatabase);

parameter(s) *podDatabase* is an input pointer to the MPE layer platform-independent POD information database used to cache the host-POD information. *mpe_PODDatabase* is described in the *mpe_PODDatabase* section.

value returned If the call is successful, *mpeos_podGetAppInfo()* should return *MPE_SUCCESS*, along with a pointer to the POD database specified by *podDatabase*. Otherwise, it should return one of the following error codes:
MPE_EINVAL indicates *podDatabase* has an invalid value.

description *mpeos_podGetAppInfo()* should identify the pointer to the POD application information. The application information is normally acquired from the host-POD interface during the initialization phase and is usually cached within the MPE layer POD database. This function is provided for platforms, where it may be necessary to take additional steps to acquire the application information. *mpeos_podGetAppInfo()* provides on-demand support if necessary.

related function(s) *mpeos_podGetFeatures*

mpeos_podGetCCIBits

gets the copy-control information for the service.

syntax mpe_Error mpeos_podGetCCIBits(
 uint32_t *frequency*,
 uint32_t *programNumber*,
 uint32_t *tsId*,
 uint8_t * *cciBits*);

parameter(s) *frequency* specifies the frequency of the transport stream on which the selected service is carried.

programNumber specifies the program number of the selected service.

tsId specifies the transport-stream identifier associated with the transport stream on which the selected service is carried.

cciBits is an output pointer where the requested Copy-Control Information (CCI) bits are returned.

value returned If the call is successful, mpeos_podGetCCIBits() should return MPE_SUCCESS, along with a pointer to the copy control information specified by *cciBits*. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

MPE_ENODATA indicates the specified service is not tuned or copy control information is not available.

description mpeos_podGetCCIBits() should get the copy-control information for the specified service.

related function(s) none

mpeos_podGetFeatures

identifies the pointer to the POD generic feature list.

syntax mpe_Error mpeos_podGetFeatures(mpe_PODDatabase * *podDatabase*);

parameter(s) *podDatabase* is an output pointer to the MPE layer platform-independent POD information database used to cache the host-POD information. *mpe_PODDatabase* is described in the *mpe_PODDatabase* section.

value returned If the call is successful, *mpeos_podGetFeatures()* should return *MPE_SUCCESS*, along with a pointer to the POD database specified by *podDatabase*. Otherwise, it should return one of the following error codes:
MPE_EINVAL indicates *podDatabase* has an invalid value.

description *mpeos_podGetFeatures()* should identify the pointer to the POD's generic features list.

related function(s) *mpeos_podGetAppInfo*

mpeos_podGetFeatureParam

populates the internal POD database with the specified feature.

syntax `mpe_Error mpeos_podGetFeatureParam(mpe_PODDatabase * podDatabase, uint32_t featureId);`

parameter(s) `podDatabase` is an input pointer to the MPE layer platform-independent POD information database used to cache the host-POD information. `mpe_PODDatabase` is described in the *mpe_PODDatabase* section.

`featureId` identifies a single feature from `mpe_PODFeatureParams`. `mpe_PODFeatureParams` is described in the *mpe_PODFeatureParams* section. Currently, the following values are supported for `featureId`:

`rf_output_channel` specifies the RF output channel data and the user interface enable/disable flag.

`pc_pin` specifies the parental control pin.

`pc_setting` specifies to the parental control channel settings.

`ippv_pin` specifies the IPPV pin.

`time_zone_offset` specifies the time-zone offset in minutes from GMT.

`daylight_savings_ctrl` specifies the daylight savings control.

`ac_outlet_ctrl` specifies the AC outlet control.

`language_ctrl` specifies the ISO 639, 3-byte language code.

`ratings_region` specifies the ratings region identifier.

`reset_pin_ctrl` specifies the reset pin control indicator.

`cable_urls_length` specifies the total size of cable URL data.

`cable_urls` specifies the cable URL.

`ea_location` specifies the EAS location code.

value returned If the call is successful, `mpeos_podGetFeatureParam()` should return `MPE_SUCCESS`, along with a pointer to the POD database specified by `podDatabase`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_podGetFeatureParam()` should populate the internal POD database with the current value of the parameter value specified by `featureId`. This value is set by a previous call to `mpeos_podSetFeatureParam()`.

related function(s) `mpeos_podSetFeatureParam`

mpeos_podMMIClose

closes a connection with the MMI resource on the POD.

syntax mpe_Error mpeos_podMMIClose(void);

parameter(s) none

value returned If the call is successful, mpeos_podMMIClose() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_ENODATA indicates no data is available for the specified request.

description mpeos_podMMIClose() should provide an optional API for systems that may require additional work to maintain the Man Machine Interface (MMI) resources when an application unregisters its MMI handler from the MMI POD application. The implementation of mpeos_podMMIClose() may need one of the following actions:

- ◆ Update internal implementation resources.
- ◆ Make an operating system function call to allow the operating system to update MMI session related resources.
- ◆ Do nothing because it is entirely possible the MMI sessions can be maintained as connected on de-registration.

related function(s) mpeos_podMMIConnect

mpeos_podMMIConnect

establishes a connection with the MMI resource on the POD.

syntax mpe_Error mpeos_podMMIConnect(void);

parameter(s) none

value returned If the call is successful, mpeos_podMMIConnect() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description mpeos_podMMIConnect() should establish a connection with the MMI resource on the POD device.

related function(s) mpeos_podMMIClose

mpeos_podReceiveAPDU

receives an APDU packet.

syntax

```
mpe_Error mpeos_podReceiveAPDU(
    uint32_t * sessionId,
    uint8_t ** apdu,
    int32 * len );
```

| | | |
|---------------------|------------------|---|
| parameter(s) | <i>sessionId</i> | is an output pointer for returning the native session handle associated with the received APDU. |
| | <i>apdu</i> | is an output pointer to the next available APDU from the POD. |
| | <i>len</i> | is an output pointer which reports the size in bytes of the APDU. |

| | |
|--------------------------|---|
| value returned | If the call is successful, <code>mpeos_podReceiveAPDU()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return one of the following error codes: |
| <code>MPE_EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| <code>MPE_ENODATA</code> | indicates the APDU received is actually the last APDU that failed to be sent. |
| <code>MPE_ENOMEM</code> | indicates the function failed due to insufficient memory resources. |

| | |
|--------------------|--|
| description | <code>mpeos_podSendAPDU()</code> should retrieve the next available APDU packet from the POD. This function should also receive “copy” APDUs which failed to be sent to the POD so that the APDU can be returned to the original sender to be notified of the failure. |
|--------------------|--|

| | |
|----------------------------|--------------------------------|
| related function(s) | <code>mpeos_podSendAPDU</code> |
|----------------------------|--------------------------------|

mpeos_podRegister

registers for pod-specific asynchronous events.

syntax `mpe_Error mpeos_podRegister(
 mpe_EventQueue queueId ,
 void * act);`

| | | |
|----------------------------|---|---|
| parameter(s) | <i>queueId</i> | identifies the queue in which to send the POD events. <i>mpe_EventQueue</i> is described in the <i>mpe_EventQueue</i> section. Currently, the following events are supported: |
| | | MPE_POD_EVENT_GF_UPDATE is sent to notify the host that a value has changed for a platform-specific feature. |
| | | MPE_POD_EVENT_APPINFO_UPDATE is sent to notify the host that a value has changed for a application feature. |
| | <i>act</i> | is an input pointer to the Asynchronous Completion Token (ACT) for the Java component that is listening for POD notifications. When the asynchronous events are sent to the queue specified in <i>queueId</i> (via <i>mpeos_eventQueueSend()</i>), the event should pass this <i>act</i> pointer in the <i>optionalEventData2</i> field. |
| value returned | If the call is successful, <i>mpeos_podRegister()</i> should return MPE_SUCCESS . Otherwise, it should return one of the following error codes: | |
| | MPE_EINVAL | indicates at least one input parameter to the function has an invalid value. |
| | MPE_ENODATA | indicates the APDU received is actually the last APDU that failed to be sent. |
| | MPE_ENOMEM | indicates the function failed due to insufficient memory resources. |
| description | <i>mpeos_podRegister()</i> should register for POD-specific asynchronous events. The platform-specific layer is responsible for notifying the stack of asynchronous changes to the POD generic features list or application information. An <i>act</i> and <i>act2</i> are provided to the native layer to facilitate this communication. | |
| related function(s) | <i>mpeos_podUnregister</i> | |

mpeos_podSASClose

provides an optional unregister from a SAS application.

syntax `mpe_Error mpeos_podSASClose(uint32_t sessionId);`

parameter(s) `sessionId` identifies the session identifier of the target SAS session.

value returned If the call is successful, `mpeos_podSASClose()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates `sessionId` has an invalid value.

description `mpeos_podSASClose()` should provide an option for systems that may require additional work to maintain session resources when an application unregisters its handler from an SAS application. The implementation of `mpeos_podSASClose()` may need to take one of the following actions:

- ◆ Update internal implementation resources.
- ◆ Make an operating system function call to allow the operating system to update session related resources.
- ◆ Do nothing because it is entirely possible the sessions can be maintained as connected on de-registration and simply reused if the same host or another host application makes a later request to connect to the SAS application.

related function(s) `mpeos_podSASConnect`

mpeos_podSASConnect

establishes a connection between a host application and a POD application.

syntax mpe_Error mpeos_podSASConnect(
 uint8_t * *appld*,
 uint16_t * *sessionId*);

parameter(s) *appld* is an input pointer to a unique identifier of the private host application.

sessionId is an output pointer to a location where the session identifier can be stored. The session identifier is established by the POD to enable further communications.

value returned If the call is successful, mpeos_podSASConnect() should return MPE_SUCCESS, along with a pointer to the POD database specified by *podDatabase*. Otherwise, it should return one of the following error codes:
MPE_EINVAL indicates *appld* has an invalid value.
MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description mpeos_podSASConnect() should establish a connection between a private host application and the corresponding POD module vendor-specific application.

related function(s) mpeos_podMMIClose

mpeos_podSendAPDU

sends an APDU packet.

syntax mpe_Error mpeos_podSendAPDU(
 uint32_t sessionId,
 uint32_t apduTag,
 uint32_t size,
 uint8_t * apdu);

parameter(s) *sessionId* identifies the native handle for the SAS session for the APDU.

apduTag specifies the APDU identifier.

size specifies the size in bytes of the data buffer portion of the APDU.

apdu is an input pointer to the APDU data buffer.

value returned If the call is successful, mpeos_podSendAPDU() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description mpeos_podSendAPDU() should send an APDU packet.

related function(s) mpeos_podReceiveAPDU

mpeos_podSetFeatureParam

sets the feature in the host-POD interface.

syntax

```
mpe_Error mpeos_podSetFeatureParam(
    uint32_t featureId,
    uint8_t * parameter,
    uint32_t size );
```

parameter(s) *featureId*

identifies a single feature from mpe_PODFeatureParams. mpe_PODFeatureParams is described in the mpe_PODFeatureParams section. Currently, the following values are supported for featureId:

rf_output_channel

specifies the RF output channel data and the user interface enable/disable flag.

pc_pin

specifies the parental control pin.

pc_setting

specifies to the parental control channel settings.

ippv_pin

specifies the IPPV pin.

time_zone_offset

specifies the time-zone offset in minutes from GMT.

daylight_savings_ctrl

specifies the daylight savings control.

ac_outlet_ctrl

specifies the AC outlet control.

language_ctrl

specifies the ISO 639, 3-byte language code.

ratings_region

specifies the ratings region identifier.

reset_pin_ctrl

specifies the reset pin control indicator.

cable_urls_length

specifies the total size of cable URL data.

cable_urls

specifies the cable URL.

ea_location

specifies the EAS location code.

parameter

is an input pointer to the value of the generic feature.

size

identifies the size in bytes of the parameter value.

value returned If the call is successful, `mpeos_podGetFeatureParam()` should return TRUE, along with a pointer to the POD database specified by `podDatabase`. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

description `mpeos_podGetFeatureParam()` should set the feature specified by `featureId` to the value specified by `parameter` in the host-POD interface.

related function(s) `mpeos_podGetFeatureParam`

mpeos_podUnregister

unregisters for POD-specific asynchronous events.

syntax mpe_Error mpeos_podUnregister(void);

parameter(s) none

value returned If the call is successful, mpeos_podUnregister() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:
MPE_EINVAL indicates the POD is not registered.

description mpeos_podUnregister() should unregister for POD-specific asynchronous events.

related function(s) mpeos_podRegister

FLT Section Filtering API

Overview

The section filtering Application Programming Interface (API) provides the ability to acquire a section, or sections, from a Moving Picture Experts Group (MPEG) table section source that meets the criteria expressed in the form of a Packet Identifier (PID) and a filtering specification.

The filter specification is modeled after the Digital Audio Visual Council (DAVIC) MPEG table section filter API. The specification consists of an arbitrarily-long positive mask/value array and an arbitrarily-long negative mask/value array.

A platform's capability to accelerate section filtering is hardware and operating system specific. It is necessary for you to convert the abstract representations of filters and filter specifications into platform-specific primitives.

Before reading this chapter, you should be:

- ◆ familiar with the MPEG table sections as defined by the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 13818-1
- ◆ familiar with `org.davic.mpeg.sections` as explained in DAVIC 1.4.1 Specification, Part 9, Appendix E.8.

After reading this chapter, you should be:

- ◆ familiar with the section filtering functionality
- ◆ familiar with the associated functional requirements your port needs to fulfill

Section filtering is designed to work very quickly. The system should make the best effort to acquire matching sections with skips.

Definitions

The following definitions, which are defined in `mpeos_filter.h` and `mpe_filterevents.h`, are used by the section filtering functions:

```
MPE_SF_INVALID_SECTION_HANDLE
MPE_SF_INVALID_UNIQUE_ID
MPE_SF_OPTION_IF_NOT_CANCELLED
MPE_SF_OPTION_RELEASE_WHEN_COMPLETE
```

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPE_SF_INVALID_SECTION_HANDLE`

`MPE_SF_INVALID_SECTION_HANDLE` represents a `mpe_FilterSectionHandle` which will never be returned by Section Filtering. It can be used to initialize any field which takes an `mpe_FilterSectionHandle`.

`MPE_SF_INVALID_SECTION_HANDLE` is defined in `mpeos_filter.h` as follows:

```
#define MPE_SF_INVALID_SECTION_HANDLE (0)
```

`MPE_SF_INVALID_UNIQUE_ID`

`MPE_SF_INVALID_UNIQUE_ID` represents a `mpe_FilterSectionHandle` which will never be returned by Section Filtering. It can be used to initialize any field which takes an `mpe_FilterSectionHandle`.

`MPE_SF_INVALID_UNIQUE_ID` is defined in `mpeos_filter.h` as follows:

```
#define MPE_SF_INVALID_UNIQUE_ID (0)
```

`MPE_SF_OPTION_IF_NOT_CANCELLED`

`MPE_SF_OPTION_IF_NOT_CANCELLED` indicates the associated operation is only carried out if the filter is in the Filter Active state.

`MPE_SF_OPTION_IF_NOT_CANCELLED` is defined in `mpeos_filter.h` as follows:

```
#define MPE_SF_OPTION_IF_NOT_CANCELLED (0x00000001)
```

`MPE_SF_OPTION_RELEASE_WHEN_COMPLETE`

`MPE_SF_OPTION_RELEASE_WHEN_COMPLETE` indicates the associated operation releases the target element after performing the operation.

`MPE_SF_OPTION_RELEASE_WHEN_COMPLETE` is defined in `mpeos_filter.h` as follows:

```
#define MPE_SF_OPTION_RELEASE_WHEN_COMPLETE (0x00000002)
```

Error codes

The following error codes, which are defined in `mpe_filtterevents.h` and `mpe_error.h`, are used by the section filtering functions:

| | |
|-------------------------------------|-------------|
| MPE_EINVAL | MPE_ENOMEM |
| MPE_SF_ERROR | |
| MPE_SF_ERROR_FILTER_NOT_AVAILABLE | |
| MPE_SF_ERROR_INVALID_SECTION_HANDLE | |
| MPE_SF_ERROR_SECTION_NOT_AVAILABLE | |
| MPE_SF_ERROR_TUNER_NOT_AT_FREQUENCY | |
| MPE_SF_ERROR_TUNER_NOT_TUNED | MPE_SUCCESS |

MPE_EINVAL

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid parameter value; `uniqueID` is invalid; `buffer`, `sectionHandle`, or `size` is `NULL`; `byteCount` is 0; `offset` is equal or greater than the section size; or an invalid `flag` is specified. The information indicated by the return of `MPE_EINVAL` is dependent on the function. `MPE_EINVAL` is defined in `mpe_error.h` as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENOMEM

`MPE_ENOMEM` indicates the function failed due to insufficient memory resources. `MPE_ENOMEM` is defined in `mpe_error.h` as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_SF_ERROR

`MPE_SF_ERROR` indicates the function failed due to an unspecified section filter error. `MPE_SF_ERROR` is defined in `mpe_filtterevents.h` as follows:

```
#define MPE_SF_ERROR (0x00002000)
```

MPE_SF_ERROR_FILTER_NOT_AVAILABLE

`MPE_SF_ERROR_FILTER_NOT_AVAILABLE` indicates one or more filters necessary for the operation are not available.

`MPE_SF_ERROR_FILTER_NOT_AVAILABLE` is defined in `mpe_filtterevents.h` as follows:

```
#define MPE_SF_ERROR_FILTER_NOT_AVAILABLE  
(MPE_SF_ERROR + 1)
```

MPE_SF_ERROR_INVALID_SECTION_HANDLE

`MPE_SF_ERROR_INVALID_SECTION_HANDLE` indicates an invalid section handle was specified. `MPE_SF_ERROR_INVALID_SECTION_HANDLE` is defined in `mpe_filtterevents.h` as follows:

```
#define MPE_SF_ERROR_INVALID_SECTION_HANDLE (MPE_SF_ERROR + 2)
```

MPE_SF_ERROR_SECTION_NOT_AVAILABLE

`MPE_SF_ERROR_SECTION_NOT_AVAILABLE` indicates the section is not available. `MPE_SF_ERROR_SECTION_NOT_AVAILABLE` is defined in `mpe_filtterevents.h` as follows:

```
#define MPE_SF_ERROR_SECTION_NOT_AVAILABLE (MPE_SF_ERROR + 3)
```

MPE_SF_ERROR_TUNER_NOT_AT_FREQUENCY

MPE_SF_ERROR_TUNER_NOT_AT_FREQUENCY indicates the tuner specified in an in-band filtering request is not at the specified frequency.

MPE_SF_ERROR_TUNER_NOT_AT_FREQUENCY is defined in `mpe_filtterevents.h` as follows:

```
#define MPE_SF_ERROR_TUNER_NOT_AT_FREQUENCY (MPE_SF_ERROR + 5)
```

MPE_SF_ERROR_TUNER_NOT_TUNED

MPE_SF_ERROR_TUNER_NOT_TUNED indicates the tuner specified in an in-band filtering request is not currently tuned. MPE_SF_ERROR_TUNER_NOT_TUNED is defined in `mpe_filtterevents.h` as follows:

```
#define MPE_SF_ERROR_TUNER_NOT_TUNED (MPE_SF_ERROR + 4)
```

MPE_SUCCESS

MPE_SUCCESS indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value, for example, zero. MPE_SUCCESS is defined in `mpe_error.h` as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h`, `mpeos_event.h`, `mpeos_filter.h`, and `mpe_types.h`, are used by the section filtering functions:

| | |
|-----------------------------------|--------------------------------------|
| <code>mpe_Error</code> | <code>mpe_EventQueue</code> |
| <code>mpe_FilterComponent</code> | <code>mpe_FilterSectionHandle</code> |
| <code>mpe_FilterSource</code> | <code>mpe_FilterSource_INB</code> |
| <code>mpe_FilterSource_00B</code> | <code>mpe_FilterSourceParams</code> |
| <code>mpe_FilterSourceType</code> | <code>mpe_FilterSpec</code> |

`mpe_Error`

`mpe_Error` specifies the type of error codes. The error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

`mpe_FilterComponent`

`mpe_FilterComponent` specifies a set of data values which can be compared against a target data array. `mpe_FilterComponent` is defined in `mpeos_filter.h` as follows:

```
typedef struct mpe_FilterComponent {
    uint32_t length;
    uint8_t * mask;
    uint8_t * values;
} mpe_FilterComponent;
```

where

`length` specifies the length, in bytes, of the mask and value arrays.

`mask` is an input pointer to an array of bytes defining the mask to be applied to the target data. The array of bytes needs to be `length` in bytes, at least.

`values` is an input pointer to an array of bytes defining the values to be compared against in the target data. The array of bytes needs to be `length` in bytes, at least.

`mpe_FilterSectionHandle`

`mpe_FilterSectionHandle` specifies a handle for a Moving Picture Experts Group (MPEG) table section. Any function returning an instance of this type should never return a value of `MPE_SF_INVALID_SECTION_HANDLE`. `mpe_FilterSectionHandle` is defined in `mpeos_filter.h` as follows:

```
typedef uint32_t mpe_FilterSectionHandle;
```

mpe_FilterSource

`mpe_FilterSource` specifies the source of sections to be filtered.
`mpe_FilterSource` is defined in `mpeos_filter.h` as follows:

```
typedef struct mpe_FilterSource {
    mpe_FilterSourceType sourceType;
    uint32_t pid;
    mpe_FilterSourceParams parm;
} mpe_FilterSource;
```

where

`sourceType` designates the type of stream source.

`mpe_FilterSourceType` is described in the *mpe_FilterSourceType* section. Currently, the following values are supported for `sourceType`:

MPE_FILTER_SOURCE_00B

specifies that the source of the MPEG table sections is an out-of-band tuner. This tuner is typically associated with a persistent low-bit rate MPEG stream containing system information tables such as the Network Information Table (NIT), Short-form Virtual Channel Table (S-VCT), and Extended Application Information Table (XAIT).

MPE_FILTER_SOURCE_INB

specifies that the source of the MPEG table sections is an in-band tuner. This tuner is typically associated with the physical channel containing the currently-selected program. The associated `source_identifier` has no designation for this source type.

MPE_FILTER_SOURCE_DSG_APPID

specifies the source of the MPEG table sections is a DOCSIS Service Gateway (DSG) application tunnel managed by section filtering.

`pid` specifies the packet identifier of the stream in which to set the filter.

`parm` contains the specific source-type parameters.
`mpe_FilterSourceParams` is described in the *mpe_FilterSourceParams* section. Currently, the following values are supported for `parm`:

`p_00B` contains parameters specific to sections acquired from an out-of-band section source.

`p_INB` contains parameters specific to sections acquired from an in-band section source.

mpe_FilterSource_INB

`mpe_FilterSource_INB` contains filter source parameters specific to an in-band/adjustable tuner. `mpe_FilterSource_INB` is defined in `mpeos_filter.h` as follows:

```
typedef struct mpe_FilterSource_INB {
    uint32_t tunerID;
    uint32_t freq;
    uint32_t tsid;
} mpe_FilterSource_INB;
```

where

`tunerID` specifies the physical tuner identifier. Supported values range from 1 to n .

`freq` specifies the frequency the `tunerID` should be tuned to in hertz.

- ◆ **NOTE:** This is used only to confirm the tuner state. Tuning of an in-band tuner needs to be performed before setting a filter.

`tsid` specifies the MPEG transport stream identifier to filter on when there is more than one transport stream per frequency. Currently, 0 is the only supported value for this parameter.

mpe_FilterSource_OOB

`mpe_FilterSource_OOB` contains filter source parameters specific to an out-of-band tuner. `mpe_FilterSource_OOB` is defined in `mpeos_filter.h` as follows:

```
typedef struct mpe_FilterSource_OOB {
    uint32_t tsid;
} mpe_FilterSource_OOB;
```

where

`tsid` designates the MPEG transport stream identifier to filter on when there is more than one transport stream. Currently, 0 is the only supported value for this parameter.

mpe_FilterSourceParams

`mpe_FilterSourceParams` contains section source parameters which are specific to the type of section source. `mpe_FilterSourceParams` is defined in `mpeos_filter.h` as follows:

```
typedef union mpe_FilterSourceParams {
    mpe_FilterSource_OOB p_OOB;
    mpe_FilterSource_INB p_INB;
} mpe_FilterSourceParams;
```

where

| | |
|--------------------|--|
| <code>p_OOB</code> | contains parameters specific to sections acquired from an out-of-band section source. <code>mpe_FilterSource_OOB</code> is described in the <i>mpe_FilterSource_OOB</i> section. |
| <code>p_INB</code> | contains parameters specific to sections acquired from an in-band section source. <code>mpe_FilterSource_INB</code> is described in the <i>mpe_FilterSource_INB</i> section. |

mpe_FilterSourceType

`mpe_FilterSourceType` specifies a type of MPEG table section source. `mpe_FilterSourceType` is defined in `mpeos_filter.h` as follows:

```
typedef enum mpe_FilterSourceType {
    MPE_FILTER_SOURCE_OOB=1,
    MPE_FILTER_SOURCE_INB,
    MPE_FILTER_SOURCE_DSG_APPID,
} mpe_FilterSourceType;
```

where

| | |
|------------------------------------|--|
| <code>MPE_FILTER_SOURCE_OOB</code> | specifies that the source of the MPEG table sections is an out-of-band tuner. This tuner is typically associated with a persistent low-bit rate MPEG stream containing system information tables such as the Network Information Table (NIT), Short-form Virtual Channel Table (S-VCT), and the Extended Application Information Table (XAIT). |
|------------------------------------|--|

`MPE_FILTER_SOURCE_INB`

specifies that the source of the MPEG table sections is an in-band tuner. This tuner is typically associated with the physical channel containing the currently-selected program. The associated `source_identifier` has no designation for this source type.

`MPE_FILTER_SOURCE_DSG_APPID`

specifies the source of the MPEG table sections is a DSG application tunnel managed by section filtering.

mpe_FilterSpec

`mpe_FilterSpec` specifies the terms that define a filter. The filter specification describes the conditional terms to be evaluated against the target MPEG table section. If the target section positive components and negative components are considered as arbitrarily-long data words, the logical expression of the filter is `((sectiondata[] & pos.mask) == pos.vals) && !((sectiondata[] & neg.mask) == neg.vals)`. The `mpe_FilterSpec` is defined in `mpeos_filter.h` as follows:

```
typedef struct mpe_FilterSpec {  
    mpe_FilterComponent pos;  
    mpe_FilterComponent neg;  
} mpe_FilterSpec;
```

where

`pos` specifies the terms that need to be present in the target data for the filter described with this `mpe_FilterSpec` to signal a match. If `pos` length equals 0, the filter is a negative-only filter. `mpe_FilterComponent` is described in the *mpe_FilterComponent* section.

`neg` specifies the terms that do not need to be present in the target data for the filter described with this `mpe_FilterSpec` to signal a match. If `neg` length equals 0, the filter is a positive-only filter. `mpe_FilterComponent` is described in the *mpe_FilterComponent* section.

Event overview

All section filtering requests are created by a call to `mpeos_filterSetFilter()`, which immediately initiates the acquisition of sections. Each event related to the request includes the unique identifier of the filtering request which originated the event. So one event queue can service many filter requests, either simultaneously or in sequence. While the machine is in the Filter Active state, acquired sections can accumulate in section filtering while the application/thread catches up.

The following diagram illustrates section filtering requests as defined by the section filtering MPEOS API:

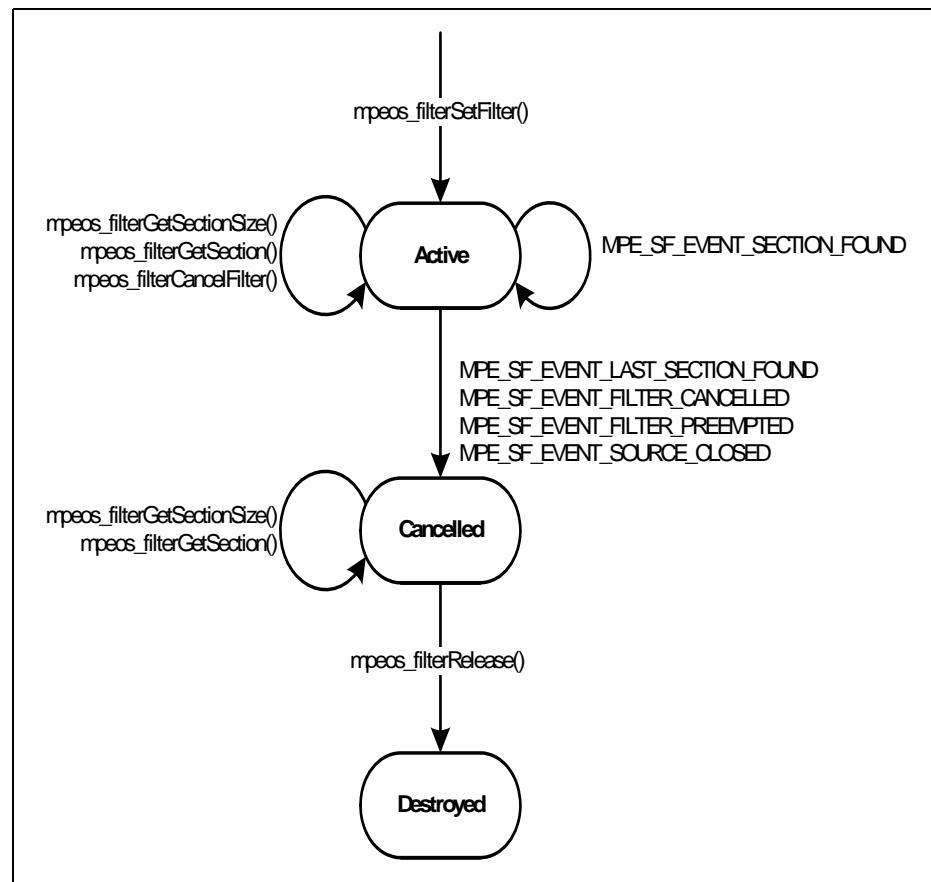


FIGURE FLT-1:
State model

The reception of a `MPE_SF_EVENT_SECTION_FOUND` or a `MPE_SF_EVENT_LAST_SECTION_FOUND` event signifies that a section has been acquired and matched on the filter. However, any sections matched by the filter are retrievable through `mpeos_filterGetSectionHandle()` regardless of the received event(s) and may be retrieved so long as the filter request is not in the Filter Released state.

`MPE_SF_EVENT_SECTION_FOUND` is the only event which may be followed by other events on the same filter. All other events indicate that the filter has been deactivated and is in the Sections Acquired state.

A call to `mpeos_filterCancelFilter()` immediately cancels filter acquisition. The client that initiates the filtering request knows that it has caught up with the cancellation upon reception of the `MPE_SF_EVENT_FILTER_CANCELLED` event.

Once a section is retrieved through `mpeos_filterGetSectionHandle()`, the section is not affected by the state of the filtering request and must be released through `mpeos_filterSectionRelease()`.

The following diagram illustrates the use of an acquired section as modeled by the section filtering MPEOS API:

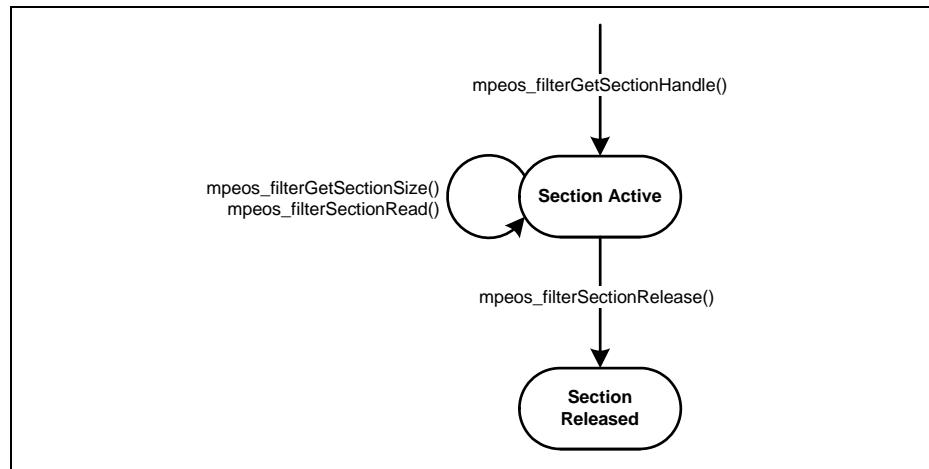


FIGURE FLT-2:
Section state model

When an event is sent (and received) it should be composed of the following information:

```

eventId = MPE_SF_FILTERS_AVAILABLE
optionalEventData1 = reserved/unused
optionalEventData2 = ACT supplied in call
    to mpeos_filterSetFilter()
optionalEventData3 = reserved/unused
    
```

Events

The following section filtering events, which are defined in `mpe_filterevents.h`, are used by the section filtering functions:

| | |
|--|--|
| <code>MPE_SF_EVENT_FILTER_AVAILABLE</code> | <code>MPE_SF_EVENT_FILTER_CANCELLED</code> |
| <code>MPE_SF_EVENT_OUT_OF_MEMORY</code> | <code>MPE_SF_EVENT_FILTER_PREEMPTED</code> |
| <code>MPE_SF_EVENT_LAST_SECTION_FOUND</code> | <code>MPE_SF_EVENT_SECTION_FOUND</code> |
| <code>MPE_SF_EVENT_SOURCE_CLOSED</code> | <code>MPE_SF_EVENT_UNKNOWN</code> |

`MPE_SF_EVENT_FILTER_AVAILABLE`

`MPE_SF_EVENT_FILTER_AVAILABLE` indicates a section filter is available for use. `MPE_SF_EVENT_FILTER_AVAILABLE` is defined as follows:

```
#define MPE_SF_EVENT_FILTER_AVAILABLE  
    (MPE_SF_EVENT_UNKNOWN + 7)
```

`MPE_SF_EVENT_FILTER_CANCELLED`

`MPE_SF_EVENT_FILTER_CANCELLED` indicates the filter was cancelled by a call to `mpeos_filterCancelFilter()`. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. `MPE_SF_EVENT_FILTER_CANCELLED` is defined as follows:

```
#define MPE_SF_EVENT_FILTER_CANCELLED  
    (MPE_SF_EVENT_UNKNOWN + 3)
```

`MPE_SF_EVENT_FILTER_PREEMPTED`

`MPE_SF_EVENT_FILTER_PREEMPTED` indicates the filter was cancelled due to a higher-priority filtering request. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. `MPE_SF_EVENT_FILTER_PREEMPTED` is defined as follows:

```
#define MPE_SF_EVENT_FILTER_PREEMPTED  
    (MPE_SF_EVENT_UNKNOWN + 4)
```

`MPE_SF_EVENT_LAST_SECTION_FOUND`

`MPE_SF_EVENT_LAST_SECTION_FOUND` indicates the last, or only, section matched by the filter is available. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. `MPE_SF_EVENT_LAST_SECTION_FOUND` is defined as follows:

```
#define MPE_SF_EVENT_LAST_SECTION_FOUND  
    (MPE_SF_EVENT_UNKNOWN + 2)
```

`MPE_SF_EVENT_OUT_OF_MEMORY`

`MPE_SF_EVENT_FILTER_OUT_OF_MEMORY` indicates filter processing has stopped because no memory is available.

`MPE_SF_EVENT_FILTER_OUT_OF_MEMORY` is defined as follows:

```
#define MPE_SF_EVENT_FILTER_OUT_OF_MEMORY  
    (MPE_SF_EVENT_UNKNOWN + 6)
```

MPE_SF_EVENT_SECTION_FOUND

MPE_SF_EVENT_SECTION_FOUND indicates a section was matched by a multi-match filter. After receiving this event, the filter remains active and may be followed by other filtering events. MPE_SF_EVENT_SECTION_FOUND is defined as follows:

```
#define MPE_SF_EVENT_SECTION_FOUND (MPE_SF_EVENT_UNKNOWN + 1)
```

MPE_SF_EVENT_SOURCE_CLOSED

MPE_SF_EVENT_SOURCE_CLOSED indicates the filter has been cancelled due to closing of the data source providing sections, or the loss of some other necessary resource. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter.

MPE_SF_EVENT_SOURCE_CLOSED is defined as follows:

```
#define MPE_SF_EVENT_SOURCE_CLOSED (MPE_SF_EVENT_UNKNOWN + 5)
```

MPE_SF_EVENT_UNKNOWN

MPE_SF_EVENT_UNKNOWN indicates an invalid event value.

MPE_SF_EVENT_UNKNOWN is defined as follows:

```
#define MPE_SF_EVENT_UNKNOWN (0x00001000)
```

Supported functions

The following section filtering functions need to be supported:

`mpeos_filterCancelFilter`
cancels the designated filter.

`mpeos_filterCloseDSGTunnel`
closes the DSG tunnel.

`mpeos_filterGetSectionHandle`
gets a handle to the section.

`mpeos_filterGetSectionSize`
gets the total size of the section.

`mpeos_filterInit`
initializes the section filtering subsystem.

`mpeos_filterOpenDSGTunnel`
opens a DSG tunnel.

`mpeos_filterRegisterAvailability`
requests the event queue be notified of additional available
section filters.

`mpeos_filterRelease`
releases the filter and associated resources.

`mpeos_filterSectionRead`
copies the section.

`mpeos_filterSectionRelease`
releases the section.

`mpeos_filterSetFilter`
sets the filter.

`mpeos_filterShutdown`
shuts down the section filtering subsystem.

`mpeos_filterUnregisterAvailability`
unregisters the event queue for available section filters.

mpeos_filterCancelFilter

cancels the designated filter.

syntax mpe_Error mpeos_filterCancelFilter(uint32_t *uniqueID*);

parameter(s) *uniqueID* identifies the filter to be cancelled. *uniqueID* is returned by a previous call to mpeos_filterSetFilter().

value returned If the call is successful, mpeos_filterCancelFilter() should return MPE_SUCCESS. Otherwise, mpeos_filterCancelFilter() should return the following error code:

MPE_EINVAL indicates *uniqueID* is invalid.

description mpeos_filterCancelFilter should cancel the designated filter. If the filter is in the Filter Active state, an MPE_SF_EVENT_FILTER_CANCELLED event is put in the event queue associated with the *uniqueID* and no further events will be sent to the queue. Any matched section data which has not been released via mpeos_filterSectionRelease() or implicitly released via mpeos_filterSectionRead() will not be released until mpeos_filterRelease() is called.

related function(s) mpeos_filterSetFilter

mpeos_filterCloseDSGTunnel

closes the DSG tunnel.

syntax mpe_Error mpeos_filterCloseDSGTunnel(void);

parameter(s) none

value returned If the call is successful, mpeos_filterCloseDSGTunnel() should return MPE_SUCCESS. Otherwise, mpeos_filterCloseDSGTunnel() should return the following error code:

MPE_SF_ERROR indicates initialization failed.

description mpeos_filterCloseDSGTunnel() should close a previously-opened DSG tunnel.

related function(s) mpeos_filterOpenDSGTunnel

mpeos_filterGetSectionHandle

gets a handle to the section.

syntax `mpe_Error mpeos_filterGetSectionHandle(`
`uint32_t uniqueID,`
`uint32_t flags,`
`mpe_FilterSectionHandle * sectionHandle);`

parameter(s) `uniqueID` identifies the filtering request. `uniqueID` is returned by a previous call to `mpeos_filterSetFilter()`.

`flags` specifies options for retrieving the handle. Currently, the value is:

`MPE_SF_OPTION_IF_NOT_CANCELLED`

indicates the associated operation is only carried out if the filter is in the Filter Active state.

`sectionHandle` is an output pointer to the destination for the section handle. The handle is used in subsequent section operations. `mpe_FilterSectionHandle` is described in the `mpe_FilterSectionHandle` section.

value returned If the call is successful, `mpeos_filterGetSectionHandle()` should return `MPE_SUCCESS`. Otherwise, `mpeos_filterGetSectionHandle()` should return one of the following error codes:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid parameter value; or `uniqueID` is invalid, `sectionHandle` is `NULL`, or an invalid flag is specified.

`MPE_ENOMEM` indicates the function failed due to insufficient memory resources.

`MPE_SF_ERROR_SECTION_NOT_AVAILABLE`

indicates the section is not available, or the `MPE_SF_OPTION_IF_NOT_CANCELLED` option is indicated and is not in the Filter Active state.

description `mpeos_filterGetSectionHandle()` should get a handle to a section matching the filtering specification set in `mpeos_filterSetFilter()`. Input parameters specified by `uniqueID` identify the section filtering request.

If the filter has matched more than one section (`mpeos_filterSetFilter()` timesToMatch is greater than 1) `mpeos_filterGetSectionHandle()` should return the sections in the order they are matched.

For efficient operation, the caller should call `mpeos_filterGetSectionHandle()` in response to an `MPE_SF_EVENT_SECTION_FOUND` or `MPE_SF_EVENT_LAST_SECTION_FOUND`.

When the caller is done accessing the section, `mpeos_filterReleaseSection()` needs to be called to release the section and the associated resources.

If MPE_SF_OPTION_IF_NOT_CANCELLED is specified as part of *flags*, a handle will be returned only if a section is available and the filter has not been cancelled. This is useful when the filter may be cancelled by another thread and the thread calling `mpeos_filterGetSectionHandle()` is not able to process sections on a cancelled filter.

related function(s)

`mpeos_filterGetSize`
`mpeos_filterSectionRead`
`mpeos_filterSectionRelease`
`mpeos_filterSetFilter`

mpeos_filterGetSize

gets the total size of the section.

syntax `mpe_Error mpeos_filterGetSize (`
 `mpe_FilterSectionHandle sectionHandle,`
 `uint32_t * size);`

parameter(s) `sectionHandle` specifies the handle assigned to the section. `sectionHandle` is returned by a previous call to `mpeos_filterGetSectionHandle()`. `mpe_FilterSectionHandle` is described in the `mpe_FilterSectionHandle` section.
`size` is an output pointer to the destination for the section size value.

value returned If the call is successful, `mpeos_filterGetSize()` should return `MPE_SUCCESS`. Otherwise, `mpeos_filterGetSize()` should return one of the following error codes:

`MPE_EINVAL` indicates `size` is `NULL`.

`MPE_SF_ERROR_INVALID_SECTION_HANDLE` indicates an invalid `sectionHandle` was specified.

description `mpeos_filterGetSize()` should get the total size in bytes of the section.

related function(s) `mpeos_filterGetSectionHandle`
`mpeos_filterSectionRead`
`mpeos_filterSectionRelease`

mpeos_filterInit

initializes the section filtering subsystem.

syntax mpe_Error mpeos_filterInit(void);

parameter(s) none

value returned If the call is successful, mpeos_filterInit() should return MPE_SUCCESS. Otherwise, mpeos_filterInit() should return the following error code:
MPE_SF_ERROR indicates initialization failed.

description mpeos_filterInit() should initialize the section filtering subsystem to prepare the subsystem for use. mpeos_filterInit() will only be called once and no other methods on this Application Programming Interface (API) will be called until this function returns with an MPE_SUCCESS return value.

related function(s) mpeos_filterSetFilter
mpeos_filterShutdown

mpeos_filterOpenDSGTunnel

opens a DSG tunnel.

syntax mpe_Error mpeos_filterOpenDSGTunnel(uint32_t appID);

parameter(s) *appID* designates the application identifier for the DSG tunnel.

value returned If the call is successful, `mpeos_filterOpenDSGTunnel()` should return `MPE_SUCCESS`. Otherwise, `mpeos_filterOpenDSGTunnel()` should return the following error code:

`MPE_SF_ERROR` indicates initialization failed.

description `mpeos_filterOpenDSGTunnel()` should initialize a DSG tunnel using *appID*.

related function(s) `mpeos_filterCloseDSGTunnel`

mpeos_filterRegisterAvailability

requests the event queue be notified of additional available section filters.

syntax mpe_Error mpeos_filterRegisterAvailability(
 mpe_EventQueue *queueId*,
 void * *ACT*);

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>queueId</i> | identifies the event queue to receive availability events. If an error is returned, no futures event are delivered. Currently, the following events are supported: |
| | | MPE_SF_EVENT_FILTER_AVAILABLE indicates a section filter is available for use. |
| | | MPE_SF_EVENT_FILTER_CANCELLED indicates the filter was cancelled by a call to mpeos_filterCancelFilter(). After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. |
| | | MPE_SF_EVENT_FILTER_PREEMPTED indicates the filter was cancelled due to a higher-priority filtering request. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. |
| | | MPE_SF_EVENT_LAST_SECTION_FOUND indicates the last, or only, section matched by the filter is available. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. |
| | | MPE_SF_EVENT_FILTER_OUT_OF_MEMORY indicates filter processing has stopped because no memory is available. |
| | | MPE_SF_EVENT_SECTION_FOUND indicates a section was matched by a multi-match filter. After receiving this event, the filter remains active and may be followed by other filtering events. |
| | | MPE_SF_EVENT_SOURCE_CLOSED indicates the filter has been cancelled due to closing of the data source providing sections, or the loss of some other necessary resource. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. |
| | | MPE_SF_EVENT_UNKNOWN indicates an invalid event value. |

| | |
|----------------------------|---|
| ACT | is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous section filtering events are sent to the event queue specified in <i>queueId</i> (via <code>mpeos_eventQueueSend()</code>), the event should pass this <i>ACT</i> pointer in the <i>optionalEventData2</i> field. |
| value returned | If the call is successful, <code>mpeos_filterRegisterAvailability()</code> should return <code>MPE_SUCCESS</code> . Otherwise, <code>mpeos_filterRegisterAvailability()</code> should return one of the following error codes: <code>MPE_EINVAL</code> indicates <i>queue</i> is invalid. <code>MPE_ENOMEM</code> indicates the function failed due to insufficient memory resources. |
| description | <code>mpeos_filterRegisterAvailability()</code> should request that the event queue specified by <i>queueId</i> be notified when section filtering resources become available. Following successful registration, a <code>MPE_SF_FILTERS_AVAILABLE</code> event will be sent to the given event queue when section filtering resources become available. On processing the notification, the client may attempt to initiate another filter request through <code>mpeos_filterSetFilter()</code> . However, the request may not succeed due to other requests being initiated in the same time frame or lack of other resources necessary for the particular request. |
| related function(s) | <code>mpeos_filterGetSectionHandle</code> <code>mpeos_filterSetFilter</code> |

mpeos_filterRelease

releases the filter and associated resources.

syntax `mpe_Error mpeos_filterRelease(uint32_t uniqueID);`

parameter(s) `uniqueID` identifies the filtering request. `uniqueID` is returned by a previous call to `mpeos_filterSetFilter()`.

value returned If the call is successful, `mpeos_filterRelease()` should return `MPE_SUCCESS`. Otherwise, `mpeos_filterRelease()` should return the following error code:

`MPE_EINVAL` indicates `uniqueID` is invalid.

description `mpeos_filterRelease()` should release the filter and any associated resources including any sections that have been matched but not yet retrieved via `mpeos_filterGetSectionHandle()`. If the filter is in the Filter Active state, an `MPE_SF_EVENT_FILTER_CANCELLED` event is put in the event queue associated with the `uniqueID` and no further events will be sent to the queue.

related function(s) `mpeos_filterGetSectionHandle`
`mpeos_filterSetFilter`

mpeos_filterSectionRead

copies the section.

syntax `mpe_Error mpeos_filterSectionRead(mpe_FilterSectionHandle sectionHandle, uint32_t offset, uint32_t byteCount, uint32_t flags, uint8_t * buffer, uint32_t * bytesRead);`

parameter(s) *sectionHandle* identifies the handle assigned to the section. *sectionHandle* is returned by a previous call to `mpeos_filterGetSectionHandle()`. `mpe_FilterSectionHandle` is described in the `mpe_FilterSectionHandle` section.

offset specifies the offset to read from within the section in bytes.

byteCount specifies the number of bytes to read from the section. This can be larger than the size of the section, for example, the target buffer size.

flags specifies options for reading the section data. The currently supported options are:

`MPE_SF_OPTION_IF_NOT_CANCELLED`

indicates the associated operation is only carried out if the filter is in the Filter Active state.

`MPE_SF_OPTION_RELEASE_WHEN_COMPLETE`

indicates the section being read will be released after being successfully read.

no option specified (*flags* == 0)

indicates the associated operation will be carried out under all filter states and will not release the section after it is read.

buffer is an output pointer to the buffer where the section data is copied.

bytesRead is an output pointer to the destination for the number of bytes actually copied.

value returned If the call is successful, `mpeos_filterSectionRead()` should return `MPE_SUCCESS`. Additionally, *bytesRead* should contain the number of bytes read. Otherwise, `mpeos_filterSectionRead()` should return one of the following error codes:

`MPE_EINVAL` is returned if *buffer* is NULL, *byteCount* is 0, *offset* is equal or greater than the section size, or an invalid flag is specified.

`MPE_SF_ERROR_INVALID_SECTION_HANDLE`

indicates an invalid *sectionHandle* was specified.

description mpeos_filterSectionRead() should copy *byteCount*, or if the number of bytes in the section is less than *byteCount*, section size minus *offset*, number of bytes starting at *offset* bytes within the section. The total number of bytes to be read should be stored in *bytesRead*, if *bytesRead* is non-NULL.

If the MPE_SF_OPTION_RELEASE_WHEN_COMPLETE option is signalled in *flags*, the section should be released on completion of a successful read and MPE_SUCCESS is returned.

related function(s) mpeos_filterGetSectionHandle
mpeos_filterGetSectionSize
mpeos_filterSectionRelease

mpeos_filterSectionRelease

releases the section.

syntax mpe_Error mpeos_filterSectionRelease (mpe_FilterSectionHandle *sectionHandle*);

parameter(s) *sectionHandle* specifies the handle for the section. *sectionHandle* is returned by a previous call to mpeos_filterSectionHandle(). *mpe_FilterSectionHandle* is described in the *mpe_FilterSectionHandle* section.

value returned If the call is successful, mpeos_filterSectionRelease() should return MPE_SUCCESS. Otherwise, mpeos_filterSectionRelease() should return the following error code:

MPE_SF_ERROR_INVALID_SECTION_HANDLE
indicates an invalid *sectionHandle* was specified.

description mpeos_filterSectionRelease() should release the section.

related function(s) mpeos_filterGetSectionHandle

mpeos_filterSetFilter

sets the filter.

```
syntax mpe_Error mpeos_filterSetFilter(
    const mpe_FilterSource * filterSource,
    const mpe_FilterSpec * filterSpec,
    mpe_EventQueue queueId,
    void * ACT,
    uint8_t filterPriority,
    uint32_t timesToMatch,
    uint32_t flags,
    uint32_t * uniqueID );
```

| | |
|---------------------|---|
| parameter(s) | <p><i>filterSource</i> is an input pointer to a description of the section source and is caller allocated. The filter operates on sections from this source. <i>mpe_FilterSource</i> is described in the <i>mpe_FilterSource</i> section.</p> <p><i>filterSpec</i> is an input pointer to the filter specification to use. This defines the criteria that the filter will evaluate sections against. <i>mpe_FilterSpec</i> is described in the <i>mpe_FilterSpec</i> section.</p> <p><i>queueId</i> identifies the queue in which to send the filter-related events. <i>mpe_EventQueue</i> is described in the <i>mpe_EventQueue</i> section. Currently, the following events are supported:</p> <ul style="list-style-type: none"> MPE_SF_EVENT_FILTER_AVAILABLE indicates a section filter is available for use. MPE_SF_EVENT_FILTER_CANCELLED indicates the filter was cancelled by a call to <i>mpeos_filterCancelFilter()</i>. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. MPE_SF_EVENT_FILTER_PREEMPTED indicates the filter was cancelled due to a higher-priority filtering request. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. MPE_SF_EVENT_LAST_SECTION_FOUND indicates the last, or only, section matched by the filter is available. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter. MPE_SF_EVENT_FILTER_OUT_OF_MEMORY indicates filter processing has stopped because no memory is available. |
|---------------------|---|

MPE_SF_EVENT_SECTION_FOUND
indicates a section was matched by a multi-match filter. After receiving this event, the filter remains active and may be followed by other filtering events.

MPE_SF_EVENT_SOURCE_CLOSED
indicates the filter has been cancelled due to closing of the data source providing sections, or the loss of some other necessary resource. After receiving this event, the filter is in the Sections Acquired State and no other events are received by the filter.

MPE_SF_EVENT_UNKNOWN
indicates an invalid event value.

ACT is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous section filtering events are sent to the event queue specified in *queueId* (via `mpeos_eventQueueSend()`), the event should pass this *ACT* pointer in the *optionalEventData2* field.

filterPriority specifies the priority of the filter with 1 being the highest, 100 being the lowest.

timesToMatch specifies the number of times the filter should match before stopping. A value of 0 indicates the filter should continue matching until cancelled.

flags specifies options for retrieving the handle. Currently no *flags* are supported for `mpeos_filterSetFilter()` and *flags* should evaluate to 0.

uniqueID is an output pointer to the unique identifier used to identify the filter in event notifications and any subsequent operations.

value returned If the call is successful, `mpeos_filterSetFilter()` should return `MPE_SUCCESS`. Otherwise, `mpeos_filterSetFilter()` should return one of the following error codes:

MPE_EINVAL indicates at least one input parameter to the function has an invalid parameter value.

MPE_ENOMEM indicates the function failed due to insufficient memory resources.

MPE_SF_ERROR_FILTER_NOT_AVAILABLE
indicates one or more filters necessary for the operation are not available.

MPE_SF_ERROR_TUNER_NOT_AT_FREQUENCY
indicates the tuner specified in an in-band filtering request is not at the specified frequency.

MPE_SF_ERROR_TUNER_NOT_TUNED
indicates the tuner specified in an in-band filtering request is not currently tuned.

description

`mpeos_filterSetFilter()` should create a filter and initiate filtering for data read from a section source, such as in `mpeos_filterSectionRead()`. `filterSpec` is a pointer to the `mpe_FilterSpec` structure which defines the filter.

`filterSpec` describes both elements which must be present (positive components) and elements which must be absent (negative components) in the section in order to match. If the target section's positive and negative components are considered arbitrarily-long data words, the logical expression of the filter is `((sectiondata[] & pos.mask) == pos.vals) && !((sectiondata[] & neg.mask) == neg.vals)`.

`filterSpec` may be de-allocated immediately after the call to `mpeos_filterSetFilter()` as `mpeos_filterSetFilter()` will process `filterSpec` before returning.

Any of the MPE_SF_EVENT events may be delivered to the queue identified by `queueHandle` during or after `mpeos_filterSetFilter()` calls until the filter is cancelled, released, or `timesToMatch` sections are found that satisfy `filterSpec`. When `timesToMatch` sections are matched, an MPE_SF_EVENT_LAST_SECTION_FOUND event is delivered to the queue. If `timesToMatch` is 0, the filter will match until cancelled or released. Setting `timesToMatch` to 0 is not recommended for high-rate streams, as sections may be retrieved faster than the caller may process them. The results could cause an over allocation of memory and other section resources. The caller should calculate an appropriate `timesToMatch` whenever possible.

The availability of filters is platform- and deployment-specific. It is understood that the Multimedia Platform Environment (MPE) is configured for the availability and underlying resource constraints, if any, associated with the filters. Priorities will be used within the MPE to provide the filtering implementation enough information to revoke filters according to the need and the ability to scale filtering needs. When filter resources are limited, the setting of a filter may be accomplished through the cancellation of a filter or filters at a lower priority. Any MPE component setting a filter with a priority greater than 1 needs to be prepared for the filter to be revoked.

-
- ◆ **NOTE:** Cancellations may occur implicitly, for example, via a tuning change when the section source is a tuner or explicitly via `mpeos_filterCancelFilter()`.
-

related function(s)

`mpeos_filterCancelFilter`
`mpeos_filterSectionRead`
`mpeos_filterShutdown`

mpeos_filterShutdown

shuts down the section filtering subsystem.

syntax mpe_Error mpeos_filterShutdown(void);

parameter(s) none

value returned If the call is successful, mpeos_filterShutdown() should return MPE_SUCCESS. Otherwise, mpeos_filterShutdown() should return the following error code:

MPE_SF_ERROR indicates shutdown failed.

description mpeos_filterShutdown() should shutdown the section filtering subsystem. mpeos_filterShutdown() is only called once. No other methods in this API will be called after this method returns, regardless of the return code. The section filtering subsystem should release as many resources as possible. Outstanding filters do not need to be issued such as MPE_SF_EVENT_SOURCE_CLOSED or any other events on shutdown.

related function(s) mpeos_filterInit

mpeos_filterUnregisterAvailability

unregisters the event queue for available section filters.

syntax mpe_Error mpeos_filterUnregisterAvailability(
 mpe_EventQueue *queueId*,
 void * *ACT*);

parameter(s) *queueId* specifies the queue in which to unregister from event notifications. *mpe_EventQueue* is described in the *mpe_EventQueue* section.

ACT is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous section filtering events are sent to the event queue specified in *queueId* (via *mpeos_eventQueueSend()*), the event should pass this *ACT* pointer in the *optionalEventData2* field.

value returned If the call is successful, *mpeos_filterUnregisterAvailability()* should return *MPE_SUCCESS*. Otherwise, *mpeos_filterUnregisterAvailability()* should return the following error code:

MPE_EINVAL indicates the *queueId* value is invalid.

description *mpeos_filterUnregisterAvailability()* should unregister the event queue and completion token pair previously created on a call to *mpeos_filterRegisterAvailability()*. After successful completion of this call, the event queue should no longer receive notifications of filter availability.

related function(s) *mpeos_filterRegisterAvailability*

SND Sound API

Overview

The Sound Application Programming Interface (API) provides support for creating and deleting players, discovering devices that can play a sound data MIME type, and controlling playback of sound data.

An engineer doing a port will need to implement port-specific definitions of three handle types:

- ◆ An `mpe_SndSound` handle represents in-memory sound data.
- ◆ An `mpe_SndDevice` handle represents a sound device on the system. It supports `mpeos_sndGetMaxPlaybacks` simultaneous playbacks. This is passed to the `mpeos_sndPlay` function.
- ◆ An `mpe_SndPlayback` handle represents an ongoing playback. It is returned by the `mpeos_sndPlay` function and used by the `mpeos_sndStop`, `mpeos_sndGetTime`, and `mpeos_sndSetTime` functions.

Before reading this chapter, you should be:

- ◆ familiar with what the sound functions are and how they work.
- ◆ familiar with the MIME types supported by your implementation.
- ◆ familiar with the devices supported by your implementation

After reading this chapter, you should be able to port the sound functionality within the OCAP stack

Error codes

The following error codes, which are defined in `mpe_error.h` and `mpeos_snd.h`, are used by the sound functions:

MPE_EINVAL
MPE_SUCCESS

MPE_ENOMEM

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

MPE_EINVAL

MPE_EINVAL indicates at least one input parameter to the function has an invalid value. MPE_EINVAL is defined in `mpe_error.h` as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENOMEM

MPE_ENOMEM indicates the function failed due to insufficient memory resource. MPE_ENOMEM is defined in `mpe_error.h` as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_SUCCESS

MPE_SUCCESS indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). MPE_SUCCESS is defined in `mpe_error.h` as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_types.h`, `mpe_ed.h`, and `mpeos_snd.h`, are used by the sound functions:

| | |
|------------------------------|---------------------------|
| <code>mpe_Bool</code> | <code>mpe_EdHandle</code> |
| <code>mpe_SndDevice</code> | <code>mpe_SndError</code> |
| <code>mpe_SndEvent</code> | <code>mpe_SndSound</code> |
| <code>mpe_SndPlayback</code> | |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_EdHandle`

`mpe_EdHandle` specifies a pointer to the handle for the event. `mpe_EdHandle` is defined in `mpe_ed.h` as follows:

```
typedef struct _mpe_EdEventPrivateInfo *mpe_EdHandle;
```

`mpe_SndDevice`

`mpe_SndDevice` specifies the handle for a sound playback device. `mpe_SndDevice` is defined in `mpeos_snd.h` as follows:

```
typedef struct { int unused; } *mpe_SndDevice;
```

`mpe_SndError`

`mpe_SndError` identifies the error codes for the sound functions. `mpe_SndError` is defined in `mpeos_snd.h` as follows:

```
typedef enum {
    MPE SND ERROR NO ERROR = MPE SUCCESS,
    MPE SND ERROR EMIME,
    MPE SND ERROR ERES
} mpe_SndError;
```

where

`MPE SND ERROR NO ERROR`
indicates successful completion of a sound function call.

`MPE SND ERROR EMIME`
indicates the (Per chapter, the first occurrence of an abbreviation is defined) MIME type is not supported.

`MPE SND ERROR ERES`
indicates the requested resources are not available.

mpe_SndEvent

`mpe_SndEvent` identifies event codes to be used. `mpe_SndEvent` is defined in `mpeos_snd.h` as follows:

```
typedef enum {
    MPE SND EVENT COMPLETE = 0x2000
    MPE SND EVENT ERROR
} mpe_SndEvent;
```

where

`MPE SND EVENT COMPLETE`
indicates a successful sound playback.

`MPE SND EVENT ERROR`
indicates an error occurred during sound playback.

mpe_SndPlayback

`mpe_SndPlayback` specifies the handle for ongoing playback on a `mpe_SndDevice`. `mpe_SndPlayback` is defined in `mpeos_snd.h` as follows:

```
typedef struct { int unused; } *mpe_SndPlayback;
```

mpe_SndSound

`mpe_SndSound` specifies the handle for an `mpe_Sound` object. `mpe_SndSound` is defined in `mpeos_snd.h` as follows:

```
typedef struct { int unused; } *mpe_SndSound;
```

Supported functions

The following sound functions need to be supported:

- `mpeos_sndCreateSound`
creates a sound object.
- `mpeos_sndDeleteSound`
deletes a sound object.
- `mpeos_sndGetDeviceCount`
gets the number of sound devices supported by the port.
- `mpeos_sndGetDevices`
gets an array of sound device handles supported by the port.
- `mpeos_sndGetDevicesForSound`
gets a sorted list of devices that can play the designated sound.
- `mpeos_sndGetMaxPlaybacks`
gets the maximum number of simultaneous playbacks supported for a device.
- `mpeos_sndGetTime`
gets the current media time in nanoseconds.
- `mpeos_sndInit`
initializes platform-specific sound support.
- `mpeos_sndPlay`
plays the sound data using the specified device.
- `mpeos_sndSetTime`
sets a new media time in nanoseconds.
- `mpeos_sndStop`
releases all resources associated with the current playback.

mpeos_sndCreateSound

creates a sound object.

syntax mpe_Error mpeos_sndCreateSound (

```
    const char * type,
    const char * data,
    uint32_t offset,
    uint32_t size,
    mpe_SndSound * sound );
```

| | | |
|---------------------|---------------|--|
| parameter(s) | <i>type</i> | is an input pointer to the MIME type of the sound data to be copied. Valid MIME types are platform specific and may include AIFF, MP3, WAV, and others. |
| | <i>data</i> | is an input pointer to the sound data to be copied. |
| | <i>offset</i> | specifies the offset in bytes from the start of the data to begin copying from. |
| | <i>size</i> | specifies the number of bytes of sound data to copy. |
| | <i>sound</i> | is an output pointer to the newly created sound handle. mpeos_sndCreateSound is required to make a copy of <i>data</i> when the player is created. mpe_SndSound is described in the <i>mpe_SndSound</i> section. |

value returned If the call is successful, mpeos_sndCreateSound() should return MPE_SUCCESS. Otherwise, it should return one the following error codes:

| | |
|---------------------|---------------------------------------|
| MPE_ENOMEM | indicates memory cannot be allocated. |
| MPE SND_ERROR_EMIME | indicates the MIME type is invalid. |

description mpeos_sndCreateSound() should create a sound object.

related function(s) mpeos_sndDeleteSound

mpeos_sndDeleteSound

Deletes a sound object.

syntax mpe_Error mpeos_sndDeleteSound (mpe_SndSound *sound*);

parameter(s) *sound* indicates the sound to delete. *mpe_SndSound* is described in the *mpe_SndSound* section.

value returned If the call is successful, *mpeos_sndDeleteSound()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE_EINVAL indicates *sound* has an invalid value.

description *mpeos_sndDeleteSound()* should delete and release the resources allocated to the *sound*.

related function(s) *mpeos_sndCreateSound*

mpeos_sndGetDeviceCount

gets the number of sound devices supported by the port.

syntax mpe_Error mpeos_sndGetDeviceCount(uint32_t * count);

parameter(s) *count* is an output pointer to the number of devices supported by the port.

value returned If the call is successful, mpeos_sndGetDeviceCount() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE EINVAL indicates *count* has an invalid value.

description mpeos_sndGetDeviceCount() should get the number of sound devices supported by the port.

related function(s) mpeos_sndGetDevices
mpeos_sndGetDevicesForSound
mpeos_sndGetMaxPlaybacks

mpeos_sndGetDevices

gets an array of sound device handles supported by the port.

syntax mpe_Error mpeos_sndGetDevices(
 mpe_SndDevice * devices,
 uint32_t * count);

parameter(s) *devices* is an output pointer to an array containing the devices supported by the port. *mpe_SndDevice* is described in the *mpe_SndDevice* section.

count is an input/output pointer. On input, it contains the number of elements in the *devices* array. On output, it is set to the number of devices assigned.

value returned If the call is successful, *mpeos_sndGetDevices()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one parameter to the function has an invalid value.

description *mpeos_sndGetDevices()* should get an array of sound device handles supported by the port.

related function(s) *mpeos_sndGetDeviceCount*
mpeos_sndGetDevicesForSound
mpeos_sndGetMaxPlaybacks

mpeos_sndGetDevicesForSound

gets a sorted list of devices that can play the designated sound.

syntax mpe_Error mpeos_sndGetDevicesForSound (
 mpe_SndSound *sound*,
 mpe_SndDevice * *devices*,
 uint32_t * *count*);

| | | |
|---------------------|----------------|--|
| parameter(s) | <i>sound</i> | specifies the sound. <i>mpe_SndSound</i> is described in the <i>mpe_SndSound</i> section. |
| | <i>devices</i> | is an output pointer to an array containing the prioritized list of devices which can play the specified sound. The list of devices and their priority are implementation-specific. <i>mpe_SndDevice</i> is described in the <i>mpe_SndDevice</i> section. |
| | <i>count</i> | is an input pointer containing the number elements in the <i>devices</i> array. On return, it indicates the number of assigned elements. |

value returned If the call is successful, *mpeos_sndGetDevicesForSound()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_sndGetDevicesForSound()* should get the devices that can play the designated sound, sorted in order of preference.

related function(s) *mpeos_sndGetDeviceCount*
mpeos_sndGetDevices
mpeos_sndGetMaxPlaybacks

mpeos_sndGetMaxPlaybacks

gets the maximum number of simultaneous playbacks supported for a device.

syntax mpe_Error mpeos_sndGetMaxPlaybacks(
 mpe_SndDevice *device*,
 int32 * *count*);

parameter(s) *device* specifies the sound device to check for simultaneous playbacks. *mpe_SndDevice* is described in the *mpe_SndDevice* section.

count is an output pointer to the number of supported simultaneous playbacks. If unlimited playbacks are supported, -1 is assigned.

value returned If the call is successful, *mpeos_sndGetMaxPlaybacks()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_sndGetMaxPlaybacks()* should get the maximum number of simultaneous playbacks supported for a device. The simultaneous playbacks on a device should never exceed this number.

related function(s) *mpeos_sndGetDeviceCount*
mpeos_sndGetDevices
mpeos_sndGetDevicesForSound

mpeos_sndGetTime

gets the current media time in nanoseconds.

syntax mpe_Error mpeos_sndGetTime(
 mpe_SndPlayback *playback*,
 int64 * *time*);

parameter(s) *playback* specifies the playback to get the time of. *mpe_SndPlayback* is described in the *mpe_SndPlayback* section.

time is an output pointer containing the current media time in nanoseconds.

value returned If the call is successful, *mpeos_sndGetTime()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_sndGetTime()* should get the current media time of the specified playback in nanoseconds from the beginning.

related function(s) *mpeos_sndPlay*
mpeos_sndSetTime
mpeos_sndStop

mpeos_sndInit

initializes platform-specific sound support.

syntax void mpeos_sndInit (void);

parameter(s) none

value returned none

description mpeos_sndInit() should initialize platform-specific sound support.

related function(s) none

mpeos_sndPlay

plays the sound data using the specified device.

syntax `mpe_Error mpeos_sndPlay (`
`mpe_SndDevice device,`
`mpe_SndSound sound,`
`mpe_EdHandle handle,`
`int64 start,`
`mpe_Bool loop,`
`mpe_SndPlayback * playback);`

| | | |
|---------------------|-----------------|--|
| parameter(s) | <i>device</i> | specifies the sound device to start playback. <code>mpe_SndDevice</code> is described in the <i>mpe_SndDevice</i> section. |
| | <i>sound</i> | specifies the sound to be played. <code>mpe_SndSound</code> is described in the <i>mpe_SndSound</i> section. |
| | <i>handle</i> | specifies the event dispatch handle to use for delivering native events back to Java. <code>mpe_EdHandle</code> is described in the <i>mpe_EdHandle</i> section. Currently, the following values are defined for delivering native events: |
| | | MPE SND EVENT COMPLETE indicates a successful sound playback. |
| | | MPE SND EVENT ERROR indicates an error occurred during sound playback. |
| | <i>start</i> | specifies the media start time in nanoseconds. |
| | <i>loop</i> | if set to TRUE, the sound is played in loop mode. If set to FALSE, the sound is only played once. <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |
| | <i>playback</i> | is an output pointer assigned to a valid playback handle. <code>mpe_SndPlayback</code> is described in the <i>mpe_SndPlayback</i> section. |

value returned If the call is successful, `mpeos_sndPlay()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

| | |
|---------------------------------|--|
| <code>MPE EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |
| <code>MPE ENOMEM</code> | indicates the memory cannot be allocated. |
| <code>MPE SND ERROR ERES</code> | indicates native resources cannot be obtained. |

description `mpeos_sndPlay()` should play the designated sound on the specified device.

related function(s) `mpeos_sndGetTime`
`mpeos_sndSetTime`
`mpeos_sndStop`

mpeos_sndSetTime

sets a new media time in nanoseconds.

syntax `mpe_Error mpeos_sndSetTime (`
 `mpe_SndPlayback playback,`
 `int64 * time);`

| | | |
|---------------------|-----------------|---|
| parameter(s) | <i>playback</i> | specifies an in-progress playback. <code>mpe_SndPlayback</code> is described in the <i>mpe_SndPlayback</i> section. |
| | <i>time</i> | is an input pointer to the media time to set. On return, it outputs the actual media time that was set (which may differ from the request). If the playback has already terminated, -1 is assigned. |

value returned If the call is successful, `mpeos_sndSetTime()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_sndSetTime()` should set the current media time in the specified playback to *time* (as the number of nanoseconds from the beginning of the data). `mpeos_sndSetTime()` returns the actual time set in the *time* variable. If playback stopped before the call was received, `mpeos_sndSetTime()` returns -1.

related function(s) `mpeos_sndGetTime`
`mpeos_sndPlay`
`mpeos_sndStop`

mpeos_sndStop

releases all resources associated with the current playback.

syntax void mpeos_sndStop (
 mpe_SndPlayback *playback*,
 int64 * *time*);

parameter(s) *playback* specifies an in-progress playback. *mpe_SndPlayback* is described in the *mpe_SndPlayback* section.

time is an output pointer reporting the media time when the stop occurred. If the playback was already stopped, -1 is assigned.

value returned If the call is successful, *mpeos_sndStop()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

 MPE EINVAL indicates at least one input parameter to the function has an invalid value.

description *mpeos_sndStop()* should release all resources associated with *playback*. *time* returns the media time in nanoseconds when the specified playback stopped.

related function(s) *mpeos_sndPlay*

STG Storage Manager API

Overview

The Storage Manager Application Programming Interface (API) primary obligation is to notify registered listeners when storage devices are added, removed, or change states. When the Storage Manager is first constructed, it has to register as a listener with the OCAP Event Dispatch Manager so that it may be notified of such events.

Before reading this chapter, you should be familiar with:

- ◆ system events
- ◆ compatible storage devices
- ◆ the OCAP system constants as defined in the OCAP specification

After reading this chapter, you should be able to port the storage functionality within the OCAP stack

Definitions

The following definitions, which are defined in `mpeos_storage.h`, are used by the storage functions:

| | | |
|-----------------------------------|---------------------------|---------------------------|
| MPE_STORAGE_MAX_DISPLAY_NAME_SIZE | MPE_STORAGE_MAX_NAME_SIZE | MPE_STORAGE_MAX_PATH_SIZE |
|-----------------------------------|---------------------------|---------------------------|

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPE_STORAGE_MAX_DISPLAY_NAME_SIZE`

`MPE_STORAGE_MAX_DISPLAY_NAME_SIZE` specifies the maximum number of characters in the display name for a storage device.

`MPE_STORAGE_MAX_DISPLAY_NAME_SIZE` is defined as follows:

```
#define MPE_STORAGE_MAX_DISPLAY_NAME_SIZE  
OS_STORAGE_MAX_DISPLAY_NAME_SIZE
```

`MPE_STORAGE_MAX_NAME_SIZE`

`MPE_STORAGE_MAX_NAME_SIZE` specifies the maximum number of characters in the storage device name. `MPE_STORAGE_MAX_NAME_SIZE` is defined as follows:

```
#define MPE_STORAGE_MAX_NAME_SIZE OS_STORAGE_MAX_NAME_SIZE
```

`MPE_STORAGE_MAX_PATH_SIZE`

`MPE_STORAGE_MAX_PATH_SIZE` specifies the maximum number of characters in the root path of a storage device. `MPE_STORAGE_MAX_PATH_SIZE` is defined as follows:

```
#define MPE_STORAGE_MAX_PATH_SIZE OS_STORAGE_MAX_PATH_SIZE
```

Data types and structures

The following data types and structures, which are defined in `mpeos_event.h`, `mpe_types.h`, and `mpeos_storage.h`, are used by the storage functions:

| | |
|---|-----------------------------------|
| <code>mpe_Bool</code> | <code>mpe_EventQueue</code> |
| <code>mpe_StorageError</code> | <code>mpe_StorageEvent</code> |
| <code>mpe_StorageHandle</code> | <code>mpe_StorageInfoParam</code> |
| <code>mpe_StorageStatus</code> | |
| <code>mpe_StorageSupportedAccessRights</code> | |
| <code>mpeos_Storage</code> | |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in the `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

`mpe_StorageError`

`mpe_StorageError` specifies the error codes for the storage API. `mpe_StorageError` is defined in the `mpeos_storage.h` as follows:

```
typedef enum {
    MPE_STORAGE_ERR_NOERR = MPE_SUCCESS,
    MPE_STORAGE_ERR_INVALID_PARAM = MPE EINVAL,
    MPE_STORAGE_ERR_OUT_OF_MEM = MPE ENOMEM,
    MPE_STORAGE_ERR_BUSY = MPE EBUSY,
    MPE_STORAGE_ERR_UNSUPPORTED,
    MPE_STORAGE_ERR_NOT_ALLOWED,
    MPE_STORAGE_ERR_DEVICE_ERR,
    MPE_STORAGE_ERR_OUT_OF_STATE,
    MPE_STORAGE_ERR_BUF_TOO_SMALL,
    MPE_STORAGE_ERR_UNKNOWN
} mpe_StorageError;
```

where

`MPE_STORAGE_ERR_NOERR`

indicates successful completion of an `mpe_` or `mpeos_` function call.

`MPE_STORAGE_ERR_INVALID_PARAM`

indicates an invalid parameter (for example, out-of-range values, null pointers, or unknown values).

`MPE_STORAGE_ERR_OUT_OF_MEM`

indicates the function failed due to insufficient memory resource.

`MPE_STORAGE_ERR_BUSY`

indicates the device or resource (for example, mutex, etc.) is busy.

MPE_STORAGE_ERR_UNSUPPORTED
 indicates the operation is not supported.

MPE_STORAGE_ERR_NOT_ALLOWED
 indicates the operation is not allowed.

MPE_STORAGE_ERR_DEVICE_ERR
 indicates a hardware device error.

MPE_STORAGE_ERR_OUT_OF_STATE
 indicates the operation not appropriate at this time. For example, if the storage manager is not initialized.

MPE_STORAGE_ERR_BUF_TOO_SMALL
 indicates the specified memory buffer is not large enough to fit all of the data.

MPE_STORAGE_ERR_UNKNOWN
 indicates an unknown error.

mpe_StorageHandle

`mpe_StorageHandle` represents a platform-independent handle or reference to a particular storage device. `mpe_StorageHandle` is defined in the `mpeos_storage.h` as follows:

```
typedef mpeos_Storage* mpe_StorageHandle;
```

mpe_StorageInfoParam

`mpe_StorageInfoParam` identifies the possible storage device attributes that may be queried via the `mpeos_storageGetInfo()`. `mpe_StorageInfoParam` is defined in the `mpeos_storage.h` as follows:

```
typedef enum {
    MPE_STORAGE_DISPLAY_NAME,
    MPE_STORAGE_NAME,
    MPE_STORAGE_FREE_SPACE,
    MPE_STORAGE_CAPACITY,
    MPE_STORAGE_STATUS,
    MPE_STORAGE_GPFS_PATH,
    MPE_STORAGE_GPFS_FREE_SPACE,
    MPE_STORAGE_MEDIAFS_PARTITION_SIZE,
    MPE_STORAGE_MEDIAFS_FREE_SPACE,
    MPE_STORAGE_SUPPORTED_ACCESS_RIGHTS
} mpe_StorageInfoParam;
```

where

MPE_STORAGE_DISPLAY_NAME
 specifies the name of the display device.

MPE_STORAGE_NAME
 specifies the name of the storage device.

MPE_STORAGE_FREE_SPACE
 specifies the total amount of free space in bytes (MEDIAFS + GPFS).

MPE_STORAGE_CAPACITY

specifies the total capacity in bytes (MEDIAFS + GPFS - HIDDENFS).

MPE_STORAGE_STATUS

specifies the status of the device. Currently, the following status values are supported:

MPE_STORAGE_STATUS_READY

indicates the device is mounted, initialized, and ready for use.

MPE_STORAGE_STATUS_OFFLINE

indicates the device is present but requires a call to `mpe_storageMakeReadyForUse()` before it can be used.

MPE_STORAGE_STATUS_BUSY

indicates the device is busy (for example, being initialized, configured, checked for consistency, or made ready to detach).

- ◆ **NOTE:** It does not indicate I/O operations are currently in progress.
-

MPE_STORAGE_STATUS_UNSUPPORTED_DEVICE

indicates the device is not supported by the platform.

MPE_STORAGE_STATUS_UNSUPPORTED_FORMAT

indicates the device type and model are supported by the platform, but the current partitioning or formatting is not supported by the platform without re-initialization and loss of the existing content.

MPE_STORAGE_STATUS_UNINITIALIZED

indicates the device is not formatted and contains no existing data.

MPE_STORAGE_STATUS_DEVICE_ERR

indicates the device is in an unrecoverable error state and cannot be used.

MPE_STORAGE_STATUS_NOT_PRESENT

indicates a detected storage device bay does not contain a removable storage device.

MPE_STORAGE_GPFS_PATH

specifies the general-purpose file-system path for file input/output.

MPE_STORAGE_GPFS_FREE_SPACE

specifies the free space in bytes on the general-purpose file system.

MPE_STORAGE_MEDIAFS_PARTITION_SIZE

specifies the size in bytes of the MEDIAFS partition.

- ◆ **NOTE:** GPFS partition size is derived by subtracting this value from the MPE_STORAGE_CAPACITY.
-

MPE_STORAGE_MEDIAFS_FREE_SPACE
specifies the size of the free space in bytes in the media file system.

MPE_STORAGE_SUPPORTED_ACCESS_RIGHTS
specifies extend file access permission access rights are supported by this storage device. Currently, the following values are supported:

- MPE_STORAGE_ACCESS_RIGHT_WORLD_READ
specifies all applications have read access.
- MPE_STORAGE_ACCESS_RIGHT_WORLD_WRITE
specifies all applications have write access.
- MPE_STORAGE_ACCESS_RIGHT_APP_READ
specifies the application has read access.
- MPE_STORAGE_ACCESS_RIGHT_APP_WRITE
specifies the application has write access.
- MPE_STORAGE_ACCESS_RIGHT_ORG_READ
specifies any application in the organization has read access.
- MPE_STORAGE_ACCESS_RIGHT_ORG_WRITE
specifies any application in the organization has write access.
- MPE_STORAGE_ACCESS_RIGHT_OTHER_READ
specifies a platform-specific read access.
- MPE_STORAGE_ACCESS_RIGHT_OTHER_WRITE
specifies a platform-specific write access.

mpe_StorageStatus mpe_StorageStatus identifies status event codes used by the Storage API. mpe_StorageStatus is defined in the `mpeos_storage.h` as follows:

```
typedef enum {
    MPE_STORAGE_STATUS_READY = 0,
    MPE_STORAGE_STATUS_OFFLINE,
    MPE_STORAGE_STATUS_BUSY,
    MPE_STORAGE_STATUS_UNSUPPORTED_DEVICE,
    MPE_STORAGE_STATUS_UNSUPPORTED_FORMAT,
    MPE_STORAGE_STATUS_UNINITIALIZED,
    MPE_STORAGE_STATUS_DEVICE_ERR,
    MPE_STORAGE_STATUS_NOT_PRESENT
} mpe_StorageStatus;
```

where

MPE_STORAGE_STATUS_READY
indicates the device is mounted, initialized, and ready for use.

MPE_STORAGE_STATUS_OFFLINE

indicates the device is present but requires a call to `mpe_storageMakeReadyForUse()` before it can be used.

MPE_STORAGE_STATUS_BUSY

indicates the device is busy (for example, being initialized, configured, checked for consistency, or made ready to detach).

- ◆ **NOTE:** It does not indicate I/O operations are currently in progress.

MPE_STORAGE_STATUS_UNSUPPORTED_DEVICE

indicates the device is not supported by the platform.

MPE_STORAGE_STATUS_UNSUPPORTED_FORMAT

indicates the device type and model are supported by the platform, but the current partitioning or formatting is not supported by the platform without re-initialization and loss of the existing content.

MPE_STORAGE_STATUS_UNINITIALIZED

indicates the device is not formatted and contains no existing data.

MPE_STORAGE_STATUS_DEVICE_ERR

indicates the device is in an unrecoverable error state and cannot be used.

MPE_STORAGE_STATUS_NOT_PRESENT

indicates a detected storage device bay does not contain a removable storage device.

`mpe_StorageSupportedAccessRights`

`mpe_StorageSupportedAccessRights` identifies the set of access rights supported by a given storage device. `mpe_StorageSupportedAccessRights` is defined in the `mpeos_storage.h` as follows:

```
typedef enum {
    MPE_STORAGE_ACCESS_RIGHT_WORLD_READ = 1,
    MPE_STORAGE_ACCESS_RIGHT_WORLD_WRITE = 2,
    MPE_STORAGE_ACCESS_RIGHT_APP_READ = 4,
    MPE_STORAGE_ACCESS_RIGHT_APP_WRITE = 8,
    MPE_STORAGE_ACCESS_RIGHT_ORG_READ = 16,
    MPE_STORAGE_ACCESS_RIGHT_ORG_WRITE = 32,
    MPE_STORAGE_ACCESS_RIGHT_OTHER_READ = 64,
    MPE_STORAGE_ACCESS_RIGHT_OTHER_WRITE = 128
} mpe_StorageSupportedAccessRights;
```

where

MPE_STORAGE_ACCESS_RIGHT_WORLD_READ

specifies all applications have read access.

MPE_STORAGE_ACCESS_RIGHT_WORLD_WRITE

specifies all applications have write access.

MPE_STORAGE_ACCESS_RIGHT_APP_READ
specifies the application has read access.

MPE_STORAGE_ACCESS_RIGHT_APP_WRITE
specifies the application has write access.

MPE_STORAGE_ACCESS_RIGHT_ORG_READ
specifies any application in the organization has read access.

MPE_STORAGE_ACCESS_RIGHT_ORG_WRITE
specifies any application in the organization has write access.

MPE_STORAGE_ACCESS_RIGHT_OTHER_READ
specifies a platform-specific read access.

MPE_STORAGE_ACCESS_RIGHT_OTHER_WRITE
specifies a platform-specific write access.

mpeos_Storage

`mpeos_Storage` specifies is an opaque data type that represents a storage device. The internal representation of this data type is known only by the platform-dependent operating system implementation. `mpeos_Storage` is defined in the `mpeos_storage.h` as follows:

```
typedef os_StorageDeviceInfo mpeos_Storage;
```

Events

The following storage events are used by the storage functions:

`mpe_StorageEvent`

`mpe_StorageEvent`

`mpe_StorageEvent` specifies the set of possible storage events triggered by `mpeos_eventQueueSend()` which identifies the event to send to the queue. The events defined in `mpe_StorageEvents` are sent to the *queueId* of `mpeos_storageRegisterQueue()`. `mpe_StorageEvent` is defined in the `mpeos_storage.h` as follows:

```
typedef enum {
    MPE_EVENT_STORAGE_ATTACHED = 0,
    MPE_EVENT_STORAGE_DETACHED,
    MPE_EVENT_STORAGE_CHANGED
} mpe_StorageEvent;
```

- ◆ **NOTE:** `mpe_StorageEvent` must remain synchronized with the event codes defined in `org.ocap.storage.StorageManagerEvent`.

where

`MPE_EVENT_STORAGE_ATTACHED`

indicates a storage device is attached.

`MPE_EVENT_STORAGE_DETACHED`

indicates the storage device was successfully made ready to detach and is incapable of being brought back on-line without physically detaching and reattaching the storage device.

`MPE_EVENT_STORAGE_CHANGED`

indicates either the storage device was successfully taken off-line and is capable of being brought back on-line without physically detaching and reattaching the device or the device has entered an unrecoverable error state.

Supported functions

The following storage functions need to be supported:

- `mpeos_storageEject`
initializes the storage device.
- `mpeos_storageGetDeviceCount`
gets the number of currently attached storage devices.
- `mpeos_storageGetDeviceList`
gets a list of storage devices.
- `mpeos_storageGetInfo`
gets information about the storage device.
- `mpeos_storageInit`
initializes the storage device.
- `mpeos_storageInitializeDevice`
initializes a storage device.
- `mpeos_storageIsDetachable`
determines whether a detachable storage device is present.
- `mpeos_storageIsDvrCapable`
determines whether the storage device can store DVR content.
- `mpeos_storageIsRemovable`
determines whether removable storage media is present.
- `mpeos_storageMakeReadyForUse`
gets the storage device ready for use.
- `mpeos_storageMakeReadyToDetach`
prepares the storage device for detaching.
- `mpeos_storageRegisterQueue`
registers a queue to receive storage events.

mpeos_storageEject

ejects the removable storage media.

syntax mpe_StorageError mpeos_storageEject(mpe_StorageHandle device);

parameter(s) *device* specifies the storage device. *device* is returned by a previous call to mpeos_storageGetDeviceList(). *mpe_StorageHandle* is described in the *mpe_StorageHandle* section.

value returned If the call is successful, mpeos_storageEject() should return MPE_STORAGE_ERR_NOERR. Otherwise, it should return the following error code:

MPE_STORAGE_ERR_INVALID_PARAM
indicates an invalid parameter.

description mpeos_storageEject() should eject the removable storage media (for example, CD, DVD, memory stick) from its bay. If this operation is not applicable to the storage device, mpeos_storageEject() should do nothing and returns success.

related function(s) mpeos_storageGetDeviceList

mpeos_storageGetDeviceCount

gets the number of currently attached storage devices.

syntax mpe_StorageError mpeos_storageGetDeviceCount(uint32_t * count);

parameter(s) *count* is an output pointer to the number of connected storage devices.

value returned If the call is successful, mpeos_storageGetDeviceCount() should return MPE_STORAGE_ERR_NOERR. Otherwise, it should return the following error code:

MPE_STORAGE_ERR_INVALID_PARAM
indicates an invalid parameter.

description mpeos_storageGetDeviceCount() should get the number of currently attached storage devices and store the value in *count*. The calling application should use this value when preallocating the storage required by mpeos_storageGetDeviceList().

related function(s) mpeos_storageGetDeviceList
mpeos_storageRegisterQueue

mpeos_storageGetDeviceList

gets a list of storage devices.

syntax `mpe_StorageError mpeos_storageGetDeviceList(uint32_t * count, mpe_StorageHandle * devices);`

| | | |
|---------------------|----------------|---|
| parameter(s) | <i>count</i> | is both an input and output pointer. On input, <i>count</i> specifies the number of device handles that can fit into the pre-allocated <i>devices</i> array passed as the second parameter. On output, <i>count</i> specifies the actual number of device handles returned in the <i>devices</i> array. |
| | <i>devices</i> | is an output pointer to the pre-allocated memory buffer for returning an array of native storage handles. <i>mpe_StorageHandle</i> is described in the <i>mpe_StorageHandle</i> section. |

value returned If the call is successful, `mpeos_storageGetDeviceList()` should return `MPE_STORAGE_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_STORAGE_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_STORAGE_ERR_BUF_TOO_SMALL`
indicates that the pre-allocated *devices* buffer is not large enough to fit all of the native device handles found on the platform. Only as many devices as can fit in the buffer are returned.

description `mpeos_storageGetDeviceList()` should get the list of currently attached or embedded storage devices. The first device in the list returned is the default storage device for the platform (for example, internal hard-disk drive).

related function(s) `mpeos_storageGetDeviceCount`

mpeos_storageGetInfo

gets information about the storage device.

syntax mpe_StorageError mpeos_storageGetInfo(
 mpe_StorageHandle *device*,
 mpe_StorageInfoParam *param*,
 void * *output*);

| | | |
|---------------------|-------------------------------------|--|
| parameter(s) | <i>device</i> | specifies the storage device. <i>device</i> is returned by a previous call to <code>mpeos_storageGetDeviceList()</code> . <code>mpe_StorageHandle</code> is described in the <i>mpe_StorageHandle</i> section. |
| | <i>param</i> | specifies the storage information parameter. <code>mpe_StorageInfoParam</code> is described in the <i>mpe_StorageInfoParam</i> section. Currently, the following values are supported for <i>param</i> : |
| | MPE_STORAGE_DISPLAY_NAME | specifies the name of the display device. |
| | MPE_STORAGE_NAME | specifies the name of the storage device. |
| | MPE_STORAGE_FREE_SPACE | specifies the total amount of free space in bytes. |
| | MPE_STORAGE_CAPACITY | specifies the total capacity in bytes. |
| | MPE_STORAGE_STATUS | specifies the status of the device. |
| | MPE_STORAGE_GPFS_PATH | specifies the general-purpose file-system path for file I/O. |
| | MPE_STORAGE_GPFS_FREE_SPACE | specifies the free space in bytes on the general-purpose file system. |
| | MPE_STORAGE_MEDIAFS_PARTITION_SIZE | specifies the size in bytes of the MEDIAFS partition. |
| | MPE_STORAGE_MEDIAFS_FREE_SPACE | specifies the size of the free space in the media file system. |
| | MPE_STORAGE_SUPPORTED_ACCESS_RIGHTS | specifies EFAP access rights are supported by this storage device. |
| | <i>output</i> | is an output pointer to value of the storage information parameter specified by <i>param</i> . |

value returned If the call is successful, `mpeos_storageGetInfo()` should return `MPE_STORAGE_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_STORAGE_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_STORAGE_ERR_OUT_OF_STATE`
indicates the operation not appropriate at this time.

description `mpeos_storageGetInfo()` should get information, specified by *param*, about the storage device specified by *device*.

related function(s) `mpeos_storageGetDeviceList`

mpeos_storagelnit

initializes the storage device.

syntax void mpeos_storageInit(void);

parameter(s) none

value returned none

description mpeos_storageInit() should initialize the storage API.
mpeos_storageInit() should be called only once per boot cycle and must
be called before any other storage functions are called. All storage
function calls made before initialization should result in an
MPE_STORAGE_ERR_OUT_OF_STATE error.

related function(s) none

mpeos_storageInitializeDevice

initializes a storage device.

syntax `mpe_StorageError mpeos_storageInitializeDevice(mpe_StorageHandle device, mpe_Bool userAuthorized, uint64_t * mediaFsSize);`

parameter(s) *device* specifies the storage device. *device* is returned by a previous call to `mpeos_storageGetDeviceList()`. `mpe_StorageHandle` is described in the *mpe_StorageHandle* section.

userAuthorized determines whether the user authorizes the request. If *value* is TRUE, the request is authorized. If *value* is FALSE, the request is not authorized. `mpe_Bool` is described in the *mpe_Bool* section.

mediaFsSize is an output pointer to the minimum size in bytes of the MEDIAFS partition requested for this device. The following are possible scenarios:

- ◆ If set to 0, no minimum MEDIAFS size is requested.
- ◆ If set to a non-NULL value, the device is repartitioned. The effects of repartitioning the device may include loss of application data on this device.
- ◆ If set to NULL, the device is formatted. The effects of formatting the device will result in loss of all stored application data.
- ◆ If the device is in the MPE_STORAGE_STATUS_READY state and NULL is specified, it reformats each partition without repartitioning the device.
- ◆ If the device is in the MPE_STORAGE_STATUS_UNINITIALIZED or MPE_STORAGE_STATUS_UNSUPPORTED_FORMAT state and NULL is specified, the device is partitioned with platform-specific default sizes for MEDIAFS and GPFS.

value returned If the call is successful, `mpeos_storageInitializeDevice()` should return `MPE_STORAGE_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_STORAGE_ERR_BUSY`
indicates the device or resource is busy.

`MPE_STORAGE_ERR_DEVICE_ERR`
indicates a hardware device error.

`MPE_STORAGE_ERR_INVALID_PARAM`
indicates an invalid parameter.

MPE_STORAGE_ERR_OUT_OF_STATE
indicates the operation not appropriate at this time.

MPE_STORAGE_ERR_UNSUPPORTED
indicates the operation is not supported.

description mpeos_storageInitializeDevice() should initializes the specified storage device for use. It is usually called when a newly attached storage device is not currently suitable for use (for example, MPE_STORAGE_STATUS_UNSUPPORTED_FORMAT or MPE_STORAGE_STATUS_UNINITIALIZED), but may also be called to reformat or repartitioned a storage device that is in the MPE_STORAGE_STATUS_READY state.

On successful invocation, the state of the device should enter the MPE_STORAGE_STATUS_BUSY state and while initialization is in progress should then enter either MPE_STORAGE_STATUS_READY or MPE_STORAGE_STATUS_DEVICE_ERR states when the operation is complete. An MPE_EVENT_STORAGE_CHANGED event should be delivered to the registered queue when the operation is initiated and again when it is complete.

related function(s) mpeos_storageGetDeviceList

mpeos_storagelsDetachable

determines whether a detachable storage device is present.

syntax mpe_StorageError mpeos_storageIsDetachable(
 mpe_StorageHandle *device*,
 mpe_Bool * *value*);

parameter(s) *device* specifies the storage device. *device* is returned by a previous call to mpeos_storageGetDeviceList(). *mpe_StorageHandle* is described in the *mpe_StorageHandle* section.

value is an output pointer value which identifies a detachable storage device is present on the corresponding bay. If *value* is TRUE, the device is present. If *value* is FALSE, the device is not present. *mpe_Bool* is described in the *mpe_Bool* section.

value returned If the call is successful, mpeos_storageIsDetachable() should return MPE_STORAGE_ERR_NOERR. Otherwise, it should return the following error code:

MPE_STORAGE_ERR_INVALID_PARAM
indicates an invalid parameter.

description mpeos_storageIsDetachable() should determine whether the detachable storage device is present on the corresponding bay.

related function(s) mpeos_storageGetDeviceList

mpeos_storagelsDvrCapable

determines whether the storage device can store DVR content.

syntax mpe_StorageError mpeos_storageIsDvrCapable(
 mpe_StorageHandle *device*,
 mpe_Bool * *value*);

parameter(s) *device* specifies the storage device. *device* is returned by a previous call to `mpeos_storageGetDeviceList()`. `mpe_StorageHandle` is described in the *mpe_StorageHandle* section.

value is an output pointer value which determines whether the storage device is DVR compatible. If *value* is TRUE, the device is capable of storing DVR content. If *value* is FALSE, the device is not capable of storing DVR content. `mpe_Bool` is described in the *mpe_Bool* section.

value returned If the call is successful, `mpeos_storageIsDvrCapable()` should return `MPE_STORAGE_ERR_NOERR`. Otherwise, it should return the following error code:

`MPE_STORAGE_ERR_INVALID_PARAM`
indicates an invalid parameter.

description `mpeos_storageIsDvrCapable()` should determine whether the storage device, specified by *device*, is capable of storing DVR content.

related function(s) none

mpeos_storagelsRemovable

determines whether removable storage media is present.

syntax mpe_StorageError mpeos_storageIsRemovable(
 mpe_StorageHandle *device*,
 mpe_Bool * *value*);

parameter(s) *device* specifies the storage device. *device* is returned by a previous call to `mpeos_storageGetDeviceList()`. `mpe_StorageHandle` is described in the *mpe_StorageHandle* section.

value is an output pointer identifying the presents of removable storage media. If *value* is TRUE, removable storage media is present. If *value* is FALSE, the removable storage media is not present. `mpe_Bool` is described in the *mpe_Bool* section.

value returned If the call is successful, `mpeos_storageIsRemovable()` should return `MPE_STORAGE_ERR_NOERR`. Otherwise, it should return the following error code:

`MPE_STORAGE_ERR_INVALID_PARAM`
indicates an invalid parameter.

description `mpeos_storageIsRemovable()` should determine whether the specified storage device contains the presence of removable storage media (for example, CD, DVD, memory stick).

related function(s) none

mpeos_storageMakeReadyForUse

gets the storage device ready for use.

syntax mpe_StorageError mpeos_storageMakeReadyForUse(
 mpe_StorageHandle *device*);

parameter(s) *device* specifies the storage device. *device* is returned by a previous call to mpeos_storageGetDeviceList(). *mpe_StorageHandle* is described in the *mpe_StorageHandle* section.

value returned If the call is successful, mpeos_storageMakeReadyForUse() should return MPE_STORAGE_ERR_NOERR. Otherwise, it should return one of the following error codes:

MPE_STORAGE_ERR_DEVICE_ERR
 indicates a hardware device error.

MPE_STORAGE_ERR_INVALID_PARAM
 indicates an invalid parameter.

MPE_STORAGE_ERR_UNSUPPORTED
 indicates the operation is not supported.

description mpeos_storageMakeReadyForUse() should make the specified detachable storage device ready to be used after having previously being made ready to detach. mpeos_storageMakeReadyForUse() will not format an unformatted device and will not reformat a device with an unsupported format. If the specified device is in the MPE_STORAGE_STATUS_OFFLINE state, mpeos_storageMakeReadyForUse() should attempt to activate the device and make it available as if it was newly attached. As a result, the device should transition to one of the following states:

MPE_STORAGE_STATUS_READY
 indicates that the device is mounted, initialized, and ready for use.

MPE_STORAGE_STATUS_UNSUPPORTED_DEVICE
 indicates the device is not supported by the platform.

MPE_STORAGE_STATUS_UNSUPPORTED_FORMAT
 indicates the device type and model are supported by the platform, but the current partitioning or formatting is not supported by the platform without re-initialization and loss of the existing contents.

MPE_STORAGE_STATUS_UNINITIALIZED
 indicates the device is not formatted and contains no existing data.

MPE_STORAGE_STATUS_DEVICE_ERR
 indicates the device is in an unrecoverable error state and cannot be used.

Upon transitioning to one of these states, an MPE_EVENT_STORAGE_CHANGED event should be delivered to the registered queue.

-
- ◆ **NOTE:** `mpeos_storageMakeReadyForUse()` is used only when detachable storage devices are supported.
-

related function(s) `mpeos_storageInitializeDevice`

mpeos_storageMakeReadyToDetach

prepares the storage device for detaching.

syntax `mpe_StorageError mpeos_storageMakeReadyToDetach(mpe_StorageHandle device);`

parameter(s) `device` specifies the storage device. `device` is returned by a previous call to `mpeos_storageGetDeviceList()`. `mpe_StorageHandle` is described in the *mpe_StorageHandle* section.

value returned If the call is successful, `mpeos_storageMakeReadyToDetach()` should return `MPE_STORAGE_ERR_NOERR`. Otherwise, it should return one of the following error codes:

`MPE_STORAGE_ERR_DEVICE_ERR`
indicates a hardware device error.

`MPE_STORAGE_ERR_INVALID_PARAM`
indicates an invalid parameter.

`MPE_STORAGE_ERR_UNSUPPORTED`
indicates the operation is not supported.

description `mpeos_storageMakeReadyToDetach()` should make the specified device ready to be safely detached. Any pending input and output operations on this device are immediately cancelled.

`mpeos_storageMakeReadyToDetach()` could result in the device transitioning to the `MPE_STORAGE_STATUS_OFFLINE` state if it is capable of being brought back on-line without physically detaching and reattaching the device. Otherwise, the device will be removed from the device list and an `MPE_EVENT_STORAGE_DETACHED` event should be delivered to the registered queue. Further input and output attempts on this device while in this state will fail until the device is reattached or brought back on-line via `mpe_storageMakeReadyForUse()`.

◆ **NOTE:** `mpeos_storageMakeReadyToDetach()` is used only when detachable storage devices are supported.

related function(s) `mpeos_storageMakeReadyForUse`

mpeos_storageRegisterQueue

registers a queue to receive storage events.

syntax mpe_StorageError mpeos_storageRegisterQueue(
 mpe_EventQueue *queueId* ,
 void * *act*);

| | | |
|----------------------------|-------------------------------|--|
| parameter(s) | <i>queueId</i> | identifies the queue in which to send the storage-related events. <i>mpe_EventQueue</i> is described in the <i>mpe_EventQueue</i> section. Currently, the following events are supported: |
| | MPE_EVENT_STORAGE_ATTACHED | indicates a storage device is attached. |
| | MPE_EVENT_STORAGE_CHANGED | indicates either the storage device was successfully taken off-line and is capable of being brought back on-line without physically detaching and reattaching the device or the device has entered an unrecoverable error state. |
| | MPE_EVENT_STORAGE_DETACHED | indicates the storage device was successfully made ready to detach and is incapable of being brought back on-line without physically detaching and reattaching the storage device. |
| <i>act</i> | | is an output pointer to an Asynchronous Completion Token (ACT). Currently, there is no ACT data associated with a storage event. |
| value returned | | If the call is successful, <i>mpeos_storageRegisterQueue()</i> should return MPE_STORAGE_ERR_NOERR. Otherwise, it should return the following error code: |
| | MPE_STORAGE_ERR_INVALID_PARAM | indicates an invalid parameter. |
| description | | <i>mpeos_storageRegisterQueue()</i> should register a specified queue to receive storage related events. Only one queue may be registered at a time. Subsequent calls replace previously registered queue. |
| related function(s) | none | |

SYN Synchronization API

Overview

The synchronization Application Programming Interface (API) provides support for management and use of synchronization constructs. The main two constructs provided are re-entrant mutexes and condition variables, which are very similar to binary semaphores.

Mutexes offer a fairly simple and optimized solution for most common synchronization tasks, while the condition variable design offers flexible and powerful set of synchronization features.

On operating systems which do not directly support condition variable primitives the condition variable implementation contained in the porting layer may be more complex than that required for mutexes.

Before reading this chapter, you should be:

- ◆ familiar with common operating system synchronization mechanisms like mutexes and semaphores
- ◆ familiar with the use of synchronization mechanisms to support Java synchronization concepts

After reading this chapter, you should be: able to port the synchronization functionality within the OCAP stack

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the synchronization functions:

| | |
|-------------------------|--------------------------|
| <code>MPE_EBUSY</code> | <code>MPE_ECOND</code> |
| <code>MPE_EINVAL</code> | <code>MPE_EMUTEX</code> |
| <code>MPE_ENOMEM</code> | <code>MPE_SUCCESS</code> |

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

`MPE_EBUSY`

`MPE_EBUSY` indicates the device or resource (for example, mutex, etc.) is busy. `MPE_EBUSY` is defined as follows:

```
#define MPE_EBUSY OS_EBUSY
```

`MPE_ECOND`

`MPE_ECOND` indicates creation/deletion/acquisition of a condition object via the function failed. `MPE_ECOND` is defined as follows:

```
#define MPE_ECOND OS_ECOND
```

`MPE_EINVAL`

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

`MPE_EMUTEX`

`MPE_EMUTEX` indicates creation/deletion/acquisition of a mutex via the function failed. `MPE_EMUTEX` is defined as follows:

```
#define MPE_EMUTEX OS_EMUTEX
```

`MPE_ENOMEM`

`MPE_ENOMEM` indicates the function failed due to insufficient memory resources. `MPE_ENOMEM` is defined as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

`MPE_SUCCESS`

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h`, `mpe_types.h`, and `mpeos_sync.h`, are used by the synchronization functions:

| | |
|------------------------|------------------------|
| <code>mpe_Bool</code> | <code>mpe_Cond</code> |
| <code>mpe_Error</code> | <code>mpe_Mutex</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Cond`

`mpe_Cond` defines the condition-type binding. `mpe_Cond` is defined in `mpeos_sync.h` as follows:

```
typedef os_Mutex mpe_Cond;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_Mutex`

`mpe_Mutex` defines mutex-type binding. `mpe_Mutex` is defined in `mpeos_sync.h` as follows:

```
typedef os_Mutex mpe_Mutex;
```

Supported functions

The following general synchronization functions need to be supported:

- `mpeos_condDelete`
destroys a condition synchronization object.
- `mpeos_condGet`
gets exclusive access to a condition object.
- `mpeos_condNew`
creates a new condition synchronization object.
- `mpeos_condSet`
sets the specified condition variable to TRUE.
- `mpeos_condUnset`
sets the specified condition variable to FALSE.
- `mpeos_condWaitFor`
gets/waits a specified amount of time for a condition variable.
- `mpeos_mutexAcquire`
acquires ownership of the target mutex.
- `mpeos_mutexAcquireTry`
non-blocking version of get/acquire/lock.
- `mpeos_mutexDelete`
destroys a mutex variable/object/structure.
- `mpeos_mutexNew`
creates a new mutex variable/object/structure.
- `mpeos_mutexRelease`
releases ownership of mutex.

mpeos_condDelete

destroys a condition synchronization object.

syntax mpe_Error mpeos_condDelete(mpe_Cond *condition*);

parameter(s) *condition* specifies the identifier for the target condition object to delete. *condition* is returned by a previous call to mpeos_condNew(). mpe_Cond is described in the *mpe_Cond* section.

value returned If the call is successful, mpeos_condDelete() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *condition* has an invalid valid.

description mpeos_condDelete() should destroy the condition synchronization object specified by *condition*.

related function(s) mpeos_condGet
mpeos_condNew
mpeos_condSet
mpeos_condUnset
mpeos_condWaitFor

mpeos_condGet

gets exclusive access to a condition object.

syntax mpe_Error mpeos_condGet(mpe_Cond *condition*);

parameter(s) *condition* specifies the identifier for the target condition object to acquire. *condition* is returned by a previous call to mpeos_condNew(). mpe_Cond is described in the *mpe_Cond* section.

value returned If the call is successful, mpeos_condGet() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *condition* has an invalid value.

description mpeos_condGet() should get exclusive access of the condition object specified by *condition*. If the condition object is in the FALSE state at the time of the call, the calling thread is suspended until the condition is set to the TRUE state. If the calling thread already has exclusive access to the condition, mpeos_condGet() should do nothing.

related function(s) mpeos_condDelete
mpeos_condNew
mpeos_condSet
mpeos_condUnset
mpeos_condWaitFor

mpeos_condNew

creates a new condition synchronization object.

syntax `mpe_Error mpeos_condNew(
 mpe_Bool autoReset,
 mpe_Bool initialState,
 mpe_Cond * condition);`

| | | |
|---------------------|---------------------|---|
| parameter(s) | <i>autoReset</i> | determines whether the new condition object automatically resets. If <i>autoReset</i> is TRUE, the condition object is automatically reset to the unset (FALSE) state when a thread acquires it using <code>mpeos_condGet()</code> . <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |
| | <i>initialState</i> | determines the initial state of the condition object. If <i>initialState</i> is TRUE, the condition object is in the set state. If <i>initialState</i> is FALSE, the condition object is in the unset state. <code>mpe_Bool</code> is described in the <i>mpe_Bool</i> section. |
| | <i>condition</i> | is an output pointer for returning the new condition identifier used for subsequent condition variable operations. <code>mpe_Cond</code> is described in the <i>mpe_Cond</i> section. |

value returned If the call is successful, `mpeos_condNew()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

| | |
|-------------------------|--|
| <code>MPE_ECOND</code> | indicates there was an internal implementation error in creating the condition variable. |
| <code>MPE_EINVAL</code> | indicates at least one input parameter to the function has an invalid value. |

description `mpeos_condNew()` should create a new condition synchronization object.

related function(s) `mpeos_condDelete`
`mpeos_condGet`
`mpeos_condSet`
`mpeos_condUnset`
`mpeos_condWaitFor`

mpeos_condSet

sets the specified condition object to TRUE.

syntax mpe_Error mpeos_condSet(mpe_Cond *condition*);

parameter(s) *condition* specifies the identifier for the target condition object to set to TRUE. *condition* is returned by a previous call to mpeos_condNew(). mpe_Cond is described in the *mpe_Cond* section.

value returned If the call is successful, mpeos_condSet() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *condition* has an invalid value.

description mpeos_condSet() should set the specified condition object to TRUE and should activate the first thread waiting.

related function(s) mpeos_condDelete
mpeos_condGet
mpeos_condNew
mpeos_condUnset
mpeos_condWaitFor

mpeos_condUnset

sets the condition object to FALSE.

syntax mpe_Error mpeos_condUnset(mpe_Cond *condition*);

parameter(s) *condition* specifies the identifier for the target condition object to set to FALSE. *condition* is returned by a previous call to mpeos_condNew(). mpe_Cond is described in the *mpe_Cond* section.

value returned If the call is successful, mpeos_condUnset() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *condition* has an invalid value.

description mpeos_condUnset() should set the condition object specified by *condition* to the FALSE state.

related function(s) mpeos_condDelete
mpeos_condGet
mpeos_condNew
mpeos_condSet
mpeos_condWaitFor

mpeos_condWaitFor

gets/waits a specified amount of time for a condition variable.

syntax `mpe_Error mpeos_condWaitFor(mpe_Cond condition, uint32_t timeout);`

| | | |
|---------------------|------------------|--|
| parameter(s) | <i>condition</i> | specifies the identifier for the target condition object to acquire. This identifier is previously returned by a call to <code>mpeos_condNew()</code> . <code>mpe_Cond</code> is described in the <i>mpe_Cond</i> section. |
| | <i>timeout</i> | specifies the maximum time in milliseconds to wait for the condition object to become TRUE. To set the timeout to wait indefinitely, set <i>timeout</i> to 0. |

value returned If the call is successful, `mpeos_condWaitFor()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

| | |
|-------------------------|--|
| <code>MPE_EBUSY</code> | indicates the specified maximum wait period expired before the condition transitioning to the TRUE state to activate the caller. |
| <code>MPE_EINVAL</code> | indicates <i>condition</i> has an invalid value. |

description `mpeos_condWaitFor()` should attempt to get exclusive access of the condition object, specified by *condition*. If the condition object is in the TRUE state at the time of the call, the calling thread is suspended for a maximum period of time as specified by the *timeout* or until the condition is set to the FALSE state. If the calling thread already has exclusive access to the condition, `mpeos_condWaitFor()` does nothing.

related function(s) `mpeos_condDelete`
`mpeos_condGet`
`mpeos_condNew`
`mpeos_condSet`
`mpeos_condUnset`

mpeos_mutexAcquire

acquires ownership of the target mutex.

syntax mpe_Error mpeos_mutexAcquire(mpe_Mutex mutex);

parameter(s) *mutex* specifies the identifier of the mutex to acquire. *mutex* is returned by a previous call to mpeos_mutexNew(). *mpe_Mutex* is described in the *mpe_Mutex* section.

value returned If the call is successful, mpeos_mutexAcquire() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

- | | |
|------------|---|
| MPE_EBUSY | indicates the mutex cannot be acquired due to ownership by another thread and the implementation cannot correctly suspend the calling thread. |
| MPE_EINVAL | indicates <i>mutex</i> has an invalid value. |
| MPE_EMUTEX | indicates there was an internal implementation error in attempting to acquire the mutex. |

description mpeos_mutexAcquire() should acquire ownership of the mutex specified by *mutex*. If another thread already owns the mutex, the calling thread should be suspended within a priority-based queue until the mutex is free for acquisition by this thread. If the mutex is already owned by the calling thread, mpeos_mutexAcquire() should return MPE_SUCCESS.

- ◆ **NOTE:** For each successful mutex acquire operation, there must be a matching mutex release operation.

related function(s) mpeos_mutexAcquireTry
mpeos_mutexDelete
mpeos_mutexNew
mpeos_mutexRelease

mpeos_mutexAcquireTry

non-blocking version of get/acquire/lock.

syntax mpe_Error mpeos_mutexAcquireTry(mpe_Mutex mutex);

parameter(s) *mutex* specifies the identifier of the mutex to acquire. *mutex* is returned by a previous call to mpeos_mutexNew(). *mpe_Mutex* is described in the *mpe_Mutex* section.

value returned If the call is successful, mpeos_mutexAcquireTry() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:
MPE_EBUSY indicates the mutex cannot be acquired due to ownership by another thread.
MPE_EINVAL indicates *mutex* has an invalid value.
MPE_EMUTEX indicates there was an internal implementation error in attempting to acquire the mutex.

description mpeos_mutexAcquireTry() is the non-blocking version of the mutex acquire. If the mutex is free, the calling thread gains exclusive ownership. And if the mutex is already owned, the calling thread is not blocked but will receive an error indicating failure to acquire the mutex. If the mutex is already owned by the calling thread, mpeos_mutexAcquireTry() should return MPE_SUCCESS.

◆ **NOTE:** For each successful mutex acquire operation, there must be a matching mutex release operation.

related function(s) mpeos_mutexAcquire
mpeos_mutexDelete
mpeos_mutexNew
mpeos_mutexRelease

mpeos_mutexDelete

destroys a mutex variable/object/structure.

syntax mpe_Error mpeos_mutexDelete(mpe_Mutex *mutex*);

parameter(s) *mutex* specifies the identifier of the mutex to destroy. *mutex* is returned by a previous call to mpeos_mutexNew(). *mpe_Mutex* is described in the *mpe_Mutex* section.

value returned If the call is successful, mpeos_mutexDelete() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates *mutex* has an invalid value.

MPE_EMUTEX indicates there was an internal implementation error in attempting to delete the mutex.

description mpeos_mutexDelete() should destroy the mutex object specified by *mutex*.

related function(s) mpeos_mutexAcquire
mpeos_mutexAcquireTry
mpeos_mutexNew
mpeos_mutexRelease

mpeos_mutexNew

creates a new mutex variable/object/structure.

syntax mpe_Error mpeos_mutexNew(mpe_Mutex * mutex);

parameter(s) *mutex* is an input pointer containing the new mutex identifier. mpe_Mutex is described in the *mpe_Mutex* section.

value returned If the call is successful, mpeos_mutexNew() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates *mutex* has an invalid value.

MPE_EMUTEX indicates there was an internal implementation error in attempting to create the mutex.

MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description mpeos_mutexNew() should create a new mutex object. It should return the identifier of the newly created in *mutex* in *mutex*.

related function(s) mpeos_mutexAcquire
mpeos_mutexAcquireTry
mpeos_mutexDelete
mpeos_mutexRelease

mpeos_mutexRelease

releases ownership of mutex.

syntax mpe_Error mpeos_mutexRelease(mpe_Mutex mutex);

parameter(s) *mutex* specifies the identifier of the mutex to release. *mutex* is returned by a previous call to mpeos_mutexNew(). *mpe_Mutex* is described in the *mpe_Mutex* section.

value returned If the call is successful, mpeos_mutexRelease() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates *mutex* has an invalid value.

MPE_EMUTEX indicates there was an internal implementation error in attempting to release the mutex.

description mpeos_mutexRelease() should release ownership of the mutex specified by *mutex*, which should activate and grant ownership to the next highest priority thread waiting on the mutex.

related function(s) mpeos_mutexAcquire
mpeos_mutexAcquireTry
mpeos_mutexDelete
mpeos_mutexNew

THR Thread API

Overview

The thread Application Programming Interface (API) is required for supporting the Java Virtual Machine (JVM) and the OpenCable Application Platform (OCAP) packages. The JVM uses threads for internal functionality (for example, garbage collection, etc.) and for support of Java application threads (for example, `java.lang.Thread`). The OCAP native will likely require threads for support of asynchronous I/O operations at a minimum.

The MPE porting layer may need to support two different asynchronous I/O models, and therefore, two different scenarios of thread use.

Before reading this chapter, you should be:

- ◆ familiar with the semantics of threads or tasks provided by most operating systems
- ◆ familiar with Java thread priority requirements

After reading this chapter, you should be able to port the thread functionality within the OCAP stack

The first model pertains to operating systems that include support for asynchronous I/O operations. These systems typically use a system thread to deliver event information to the target process/thread. In this case, MPE may or may not need to explicitly create an additional thread(s) to deliver events to Java. It may be the case that the event delivery to the Java side is completely handled by the Java event queue thread.

In the second model, the operating system does not have primitive support for all or some of the asynchronous I/O operations where OCAP specifies asynchronous I/O capability. In this case, MPE may need to have a thread pool available to perform these blocking I/O operations on behalf of the caller and subsequently deliver the I/O completion events to the Java event queues.

The internal porting layer implementation for threads may need to maintain a small data structure associated with each thread. This thread descriptor may be necessary to hold thread specific data for the thread specific data functions, as well as, other thread specific information required for the implementation and maintenance of threads within the MPEOS porting layer.

Definitions

The following definitions, which are defined in `mpeos_thread.h`, are used by the thread functions:

| | |
|--|---|
| <code>MPE_THREAD_PRIOR_DFLT</code> | <code>MPE_THREAD_PRIOR_INC</code> |
| <code>MPE_THREAD_PRIOR_MAX</code> | <code>MPE_THREAD_PRIOR_MIN</code> |
| <code>MPE_THREAD_PRIOR_SYSTEM</code> | <code>MPE_THREAD_PRIOR_SYSTEM_HI</code> |
| <code>MPE_THREAD_PRIOR_SYSTEM_MED</code> | <code>MPE_THREAD_STACK_SIZE</code> |
| <code>MPE_THREAD_STAT_CALLB</code> | <code>MPE_THREAD_STAT_COND</code> |
| <code>MPE_THREAD_STAT_DEATH</code> | <code>MPE_THREAD_STAT_EVENT</code> |
| <code>MPE_THREAD_STAT_MUTEX</code> | |

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

`MPE_THREAD_PRIOR_DFLT`

`MPE_THREAD_PRIOR_DFLT` specifies the default priority for threads. `MPE_THREAD_PRIOR_DFLT` is defined as follows:

```
#define MPE_THREAD_PRIOR_DFLT OS_THREAD_PRIOR_DFLT
```

`MPE_THREAD_PRIOR_INC`

`MPE_THREAD_PRIOR_INC` specifies thread priority as one increment from the last priority (maximum, minimum, or default). `MPE_THREAD_PRIOR_INC` is defined as follows:

```
#define MPE_THREAD_PRIOR_INC OS_THREAD_PRIOR_INC
```

`MPE_THREAD_PRIOR_MAX`

`MPE_THREAD_PRIOR_MAX` specifies the maximum (highest) thread priority. `MPE_THREAD_PRIOR_MAX` is defined as follows:

```
#define MPE_THREAD_PRIOR_MAX OS_THREAD_PRIOR_MAX
```

`MPE_THREAD_PRIOR_MIN`

`MPE_THREAD_PRIOR_MIN` specifies the minimum (lowest) thread priority. `MPE_THREAD_PRIOR_MIN` is defined as follows:

```
#define MPE_THREAD_PRIOR_MIN OS_THREAD_PRIOR_MIN
```

`MPE_THREAD_PRIOR_SYSTEM`

`MPE_THREAD_PRIOR_SYSTEM_MED` specifies the lowest (minimum) priority for the system implementation thread. `MPE_THREAD_PRIOR_SYSTEM` is defined as follows:

```
#define MPE_THREAD_PRIOR_SYSTEM OS_THREAD_PRIOR_SYSTEM
```

MPE_THREAD_PRIOR_SYSTEM_HI

MPE_THREAD_PRIOR_SYSTEM_HI specifies a highest (maximum) priority for the system implementation thread. MPE_THREAD_PRIOR_SYSTEM_HI is defined as follows:

```
#define MPE_THREAD_PRIOR_SYSTEM_HI OS_THREAD_PRIOR_SYSTEM_HI
```

MPE_THREAD_PRIOR_SYSTEM_MED

MPE_THREAD_PRIOR_SYSTEM_MED specifies a medium (moderate) priority for the system implementation thread. MPE_THREAD_PRIOR_SYSTEM_MED is defined as follows:

```
#define MPE_THREAD_PRIOR_SYSTEM_MED OS_THREAD_PRIOR_SYSTEM_MED
```

MPE_THREAD_STACK_SIZE

MPE_THREAD_STACK_SIZE specifies the default stack size.

MPE_THREAD_STACK_SIZE is defined as follows:

```
#define MPE_THREAD_STACK_SIZE OS_THREAD_STACK_SIZE
```

MPE_THREAD_STAT_CALLB

MPE_THREAD_STAT_CALLB specifies the thread status is event callback context mode. MPE_THREAD_STAT_CALLB is defined as follows:

```
#define MPE_THREAD_STAT_CALLB (0x00000008)
```

MPE_THREAD_STAT_COND

MPE_THREAD_STAT_COND specifies the thread status is acquire condition or acquired condition mode. MPE_THREAD_STAT_COND is defined as follows:

```
#define MPE_THREAD_STAT_COND (0x00000002)
```

MPE_THREAD_STAT_DEATH

MPE_THREAD_STAT_DEATH specifies the thread status is thread death mode. MPE_THREAD_STAT_DEATH is defined as follows:

```
#define MPE_THREAD_STAT_DEATH (0x00008000)
```

MPE_THREAD_STAT_EVENT

MPE_THREAD_STAT_EVENT specifies the thread status is get event mode.

MPE_THREAD_STAT_EVENT is defined as follows:

```
#define MPE_THREAD_STAT_EVENT (0x00000004)
```

MPE_THREAD_STAT_MUTEX

MPE_THREAD_STAT_MUTEX specifies the thread status is acquire mutex or acquired mutex mode. MPE_THREAD_STAT_MUTEX is defined as follows:

```
#define MPE_THREAD_STAT_MUTEX (0x00000001)
```

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the thread functions:

`MPE_EINVAL`
`MPE_SUCCESS`

`MPE_ENOMEM`

MPE_EINVAL

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENOMEM

`MPE_ENOMEM` indicates the function failed due to insufficient memory resource. `MPE_ENOMEM` is defined as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_SUCCESS

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h` and `mpeos_thread.h`, are used by the thread functions:

| | |
|------------------------------------|-----------------------------|
| <code>mpe_Error</code> | <code>mpe_ThreadID</code> |
| <code>mpe_ThreadPrivateData</code> | <code>mpe_ThreadStat</code> |

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_ThreadID`

`mpe_ThreadID` specifies the thread identifier type binding. `mpe_ThreadID` is defined in `mpeos_thread.h` as follows:

```
typedef os_ThreadID mpe_ThreadID;
```

`mpe_ThreadPrivateData`

`mpe_ThreadPrivateData` specifies the private data for threads. `mpe_ThreadPrivateData` is defined in `mpeos_thread.h` as follows:

```
typedef struct mpe_ThreadPrivateData { void* threadData;
                                         int64 memCallbackId;
                                         } mpe_ThreadPrivateData;
```

where

`threadData` is a pointer to the target thread data. `threadData` is returned by a call to `mpe_threadGetData()`.

`memCallbackId`

specifies contextual information for resource reclamation for memory management. This value is inherited from parent.

`mpe_ThreadStat`

`mpe_ThreadStat` specifies the thread status type. `mpe_ThreadStat` is defined in `mpeos_thread.h` as follows:

```
typedef os_ThreadStat mpe_ThreadStat;
```

Supported functions

The following functions need to be supported for thread support:

- `mpeos_threadAttach`
registers/unregisters a thread not previously launched.
- `mpeos_threadCreate`
creates a new thread.
- `mpeos_threadDestroy`
terminates the current thread.
- `mpeos_threadGetCurrent`
gets the identifier of the current thread.
- `mpeos_threadGetData`
gets the target thread's local storage.
- `mpeos_threadGetPrivateData`
gets the MPE private data for the designated thread.
- `mpeos_threadGetStatus`
gets the target thread's status.
- `mpeos_threadSetData`
sets the target thread's local storage.
- `mpeos_threadSetPriority`
sets the target thread's priority.
- `mpeos_threadsetStatus`
sets the target thread's status variable.
- `mpeos_threadSleep`
puts current thread to sleep for specified time.
- `mpeos_threadYield`
releases the remainder of the calling thread's current time-slice.

mpeos_threadAttach

registers/unregisters a thread not previously launched.

syntax mpe_Error mpeos_threadAttach(mpe_ThreadId * *threadId*);

parameter(s) *threadId* is an output pointer for returning the target thread identifier. *mpe_ThreadId* is described in the *mpe_ThreadId* section.

value returned If the call is successful, *mpeos_threadAttach()* should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description *mpeos_threadAttach()* should create or release thread tracking information for a thread not created with *mpeos_threadCreate()*. Most likely, *mpeos_threadAttach()* should be used only for the initial primordial thread of the JVM, but there may be other situations where a native thread may need to be attached to the JVM via the Java Native Interface (JNI), which will result in a call to *mpeos_threadAttach()*.

If *threadId* is NULL, the call is considered a detach call and all resources established during the attach phase are released.

related function(s) *mpeos_threadCreate*
mpeos_threadDestroy

mpeos_threadCreate

creates a new thread.

syntax mpe_Error mpeos_threadCreate(
 void (* entry) (void *),
 void * data,
 uint32_t priority,
 unit32 stackSize,
 mpe_ThreadId * threadId,
 const char * name);

| | | |
|---------------------|-----------------------|--|
| parameter(s) | <i>entry</i> | is an input function pointer for the thread's execution entry point. The function pointer definition specifies receiving a single parameter, which is a void pointer to any data the thread requires. |
| | <i>data</i> | is an input pointer to thread-specific data being passed as the single parameter to the thread's execution entry point. |
| | <i>priority</i> | specifies the initial execution priority of the thread. The JVM assumes ten logical priority levels are available for threads. The JVM porting layer will always express priority as a number from 1 to 10, with 10 being the highest priority. The MPE should provide macros or another mechanism (such as an indexed array) to map the logical levels to target-specific priority levels. Additional priorities should be provided for threads used within the native implementation. For these threads, valid values for <i>priority</i> are: |
| | MPE_THREAD_PRIOR_DFLT | specifies the default thread priority. |
| | MPE_THREAD_PRIOR_MAX | specifies the maximum (highest) thread priority. |
| | MPE_THREAD_PRIOR_MIN | specifies the minimum (lowest) thread priority. |
| | also | |
| | MPE_THREAD_PRIOR_INC | may be added to the above values to increase the thread priority from MPE_THREAD_PRIOR_DFLT or MPE_THREAD_PRIOR_MIN, or subtracted from the above values to decrease the thread priority from MPE_THREAD_PRIOR_DFLT or MPE_THREAD_PRIOR_MAX. |
| | <i>stackSize</i> | specifies the size of the new thread stack memory. |
| | <i>threadId</i> | is an output pointer to the target thread identifier. <i>mpe_ThreadId</i> is described in the <i>mpe_ThreadId</i> section. |

name is an input pointer to the name of the new thread. Actual support for thread naming is optional within the implementation. If the target operating system does not support thread naming, there is no hard requirement for the MPE implementation to support this feature. It is purely a convenience feature, which can be useful for debugging purposes. Most JVM support thread naming internally, and hence, do not require support from the operating system.

value returned If the call is successful, `mpeos_threadCreate()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

- `MPE_EINVAL` indicates at least one input parameter to the function has an invalid value
- `MPE_ENOMEM` indicates the function failed due to insufficient memory resources.

description `mpeos_threadCreate()` should create a new operating system-specific thread. If the target operating system allows it, the new thread's priority should be set to the value specified by *priority*.

related function(s) `mpeos_threadAttach`
`mpeos_threadDestroy`

mpeos_threadDestroy

terminates the current thread.

syntax `mpe_Error mpeos_threadDestroy(mpe_ThreadId threadId);`

parameter(s) `threadId` specifies the identifier of the target thread to terminate. If `threadId` is 0 or matches the identifier of the current thread, the calling thread is terminated. `threadId` is returned from a previous call to `mpeos_threadCreate()`. `mpe_ThreadId` is described in the `mpe_Thread` section.

value returned If the call is successful, `mpeos_threadDestroy()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE EINVAL` indicates `threadId` has an invalid value or the operating system does not support arbitrary thread termination.

description `mpeos_threadDestroy()` terminates the current thread or a specified target operating system thread.

related function(s) `mpeos_threadAttach`
`mpeos_threadCreate`

mpeos_threadGetCurrent

gets the identifier of the current thread.

syntax mpe_Error mpeos_threadGetCurrent(mpe_ThreadId * *threadId*);

parameter(s) *threadId* is an output pointer for returning the thread identifier of the currently executing thread. *mpe_ThreadId* is described in the *mpe_ThreadId* section.

value returned If the call is successful, *mpeos_threadGetCurrent()* should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE EINVAL indicates *threadId* has an invalid value.

description *mpeos_threadGetCurrent()* should get the thread identifier of the current (calling) thread.

related function(s) *mpeos_threadCreate*

mpeos_threadGetData

gets the target thread's local storage.

syntax mpe_Error mpeos_threadGetData(
 mpe_ThreadId *threadId*,
 void ** *threadLocals*);

parameter(s) *threadId* specifies the identifier of the target thread whose thread locals are to be acquired. If *threadId* is 0 or matches the identifier of the current thread, calling thread's "thread local storage" is returned. *threadId* is returned from a previous call to mpeos_threadCreate(). mpe_ThreadId is described in the *mpe_ThreadId* section.
threadLocals is an output pointer used to return the specified thread's local storage pointer.

value returned If the call is successful, mpeos_threadGetData() should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *threadId* has an invalid value.

description mpeos_threadGetData() should get the thread local storage pointer for the specified thread. The *threadLocals* for Java threads should be the JVM internal thread data pointer.

related function(s) mpeos_threadGetPrivateData
 mpeos_threadSetData

mpeos_threadGetPrivateData

gets the MPE private data for the designated thread.

syntax mpe_Error mpeos_threadGetPrivateData(
 mpe_ThreadId *threadId*,
 mpe_ThreadPrivateData ** *data*);

parameter(s) *threadId* specifies the thread whose data should be returned. If *threadId* evaluates to NULL or 0 data for the current thread is returned. *mpe_ThreadId* is described in the *mpe_ThreadId* section.

data is an output pointer used to return the specified thread's private data pointer. *mpe_ThreadPrivateData* is described in the *mpe_ThreadPrivateData* section.

value returned If the call is successful, *mpeos_threadGetPrivateData()* should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE_EINVAL indicates *threadId* has an invalid value.

description *mpeos_threadGetPrivateData()* should get pointer to the MPE private data for the designated thread.

related function(s) none

mpeos_threadGetStatus

gets the target thread's status.

syntax `mpe_Error mpeos_threadGetStatus(
 mpe_ThreadId threadId,
 uint32_t * threadStatus);`

parameter(s) *threadId* specifies the identifier of the target thread. If *threadId* is 0 or matches the identifier of the current thread, the status of the calling thread is returned. *threadId* is returned from a previous call to `mpeos_threadCreate()`. `mpe_ThreadId` is described in the *mpe_ThreadId* section.

threadStatus is an output pointer for returning the current status of the thread. Currently, the following values are supported for *threadStatus*:

`MPE_THREAD_STAT_CALLB`

specifies the thread status is event callback context mode.

`MPE_THREAD_STAT_COND`

specifies the thread status is acquire condition or acquired condition mode.

`MPE_THREAD_STAT_DEATH`

specifies the thread status is thread death mode.

`MPE_THREAD_STAT_EVENT`

specifies the thread status is get event mode.

`MPE_THREAD_STAT_MUTEX`

specifies the thread status is acquire mutex or acquired mutex mode.

value returned If the call is successful, `mpeos_threadGetStatus()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates *threadId* has an invalid value.

description `mpeos_threadGetStatus()` should get the status variable of the thread specified by *threadId*.

related function(s) `mpeos_threadCreate`
`mpeos_threadSetStatus`

mpeos_threadSetData

sets the target thread's local storage.

syntax `mpe_Error mpeos_threadSetData(mpe_ThreadId threadId, void * threadLocals);`

parameter(s) *threadId* specifies the identifier of the thread whose locals are to be set. If *threadId* is 0 or matches the identifier of the current thread, the thread locals for the calling thread is set. *threadId* is returned from a previous call to `mpeos_threadCreate()`. `mpe_ThreadId` is described in the *mpe_ThreadId* section.

threadLocals is an input pointer to thread-specific data.

value returned If the call is successful, `mpeos_threadSetData()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_threadSetData()` should set the “thread local storage” for the thread specified by *threadId*. This is a pointer to a data area defined by MPE to support threads within the MPE code space and the JVM.

related function(s) `mpeos_threadCreate`
`mpeos_threadGetData`

mpeos_threadSetPriority

sets the target thread's priority.

syntax `mpe_Error mpeos_threadSetPriority(
 mpe_ThreadId threadId,
 uint32_t priority);`

| | | |
|---------------------|------------------------------------|--|
| parameter(s) | <i>threadId</i> | specifies the identifier of the target thread. <i>threadId</i> is returned from a previous call to <code>mpeos_threadCreate()</code> . <code>mpe_ThreadId</code> is described in the <i>mpe_ThreadId</i> section. |
| | <i>priority</i> | specifies the new priority of the target thread if allowed by the implementation. The JVM assumes ten logical priority levels are available for threads. The JVM porting layer will always express priority as a number from 1 to 10, with 10 being the highest priority. The MPE should provide macros or another mechanism (such as an indexed array) to map the logical levels to target-specific priority levels. Additional priorities should be provided for threads used within the native implementation. For these threads, valid values for <i>priority</i> are: |
| | <code>MPE_THREAD_PRIOR_DFLT</code> | specifies the default thread priority. |
| | <code>MPE_THREAD_PRIOR_INC</code> | used to modify the thread priority from default, maximum, or minimum. |
| | <code>MPE_THREAD_PRIOR_MAX</code> | specifies the maximum (highest) thread priority. |
| | <code>MPE_THREAD_PRIOR_MIN</code> | specifies the minimum (lowest) thread priority. |

value returned If the call is successful, `mpeos_threadSetPriority()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_threadSetPriority()` should set the priority of the thread specified by *threadId* to the value specified by *priority*.

If the target operating system does not support modifying the priority of a thread after the thread is created, then the call to `mpeos_threadSetPriority()` has no effect.

- ◆ **NOTE:** If *priority* is set for a thread, the thread must keep track of its own priority. No mechanism is provided for checking a thread's priority.

related function(s) `mpeos_threadCreate`

mpeos_threadSetStatus

sets the target thread's status variable.

syntax `mpe_Error mpeos_threadSetStatus(
 mpe_ThreadId threadId,
 uint32_t threadStatus);`

parameter(s) *threadId* specifies the identifier of the target thread. If *threadId* is 0 or matches the identifier of the current thread, the calling thread's status is set. *threadId* is returned from a previous call to `mpeos_threadCreate()`. `mpe_ThreadId` is described in the *mpe_ThreadId* section.

threadStatus specifies the new status to use for the thread. Currently, the following values are supported for *threadStatus*:

`MPE_THREAD_STAT_CALLB`
specifies the status is event callback context mode.

`MPE_THREAD_STAT_COND`
specifies the status is acquire condition or acquired condition mode.

`MPE_THREAD_STAT_DEATH`
specifies the status is thread death mode.

`MPE_THREAD_STAT_EVENT`
specifies the status is get event mode.

`MPE_THREAD_STAT_MUTEX`
specifies the status is acquire mutex or acquired mutex mode.

value returned If the call is successful, `mpeos_threadSetStatus()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_threadSetStatus()` should set the status of the thread specified by *threadId* to the value specified by *threadStatus*.

related function(s) `mpeos_threadGetStatus`
`mpeos_threadCreate`

mpeos_threadSleep

puts current thread to sleep for specified time.

syntax mpe_Error mpeos_threadSleep(
 uint32_t *milliseconds*,
 uint32_t *microseconds*);

parameter(s) *milliseconds* is the number of milliseconds to sleep.
 microseconds is the number of microseconds to sleep.

value returned If the call is successful, mpeos_threadSleep() should return MPE_SUCCESS. If the sleep could not be performed for a platform specific reason, it could return a platform specific error.

description mpeos_threadSleep() should put the current thread to sleep (that is, suspend execution) for the specified time. The amount of time the thread will be suspended is the sum of the number of milliseconds and the number of microseconds specified.

related function(s) mpeos_threadCreate

mpeos_threadYield

releases the remainder of the calling thread's current time-slice.

syntax void mpeos_threadYield(void);

parameter(s) none

value returned none

description mpeos_threadYield() should give up the remainder of the calling thread's current time-slice and be rescheduled among the other active threads currently executing in the system.

related function(s) mpeos_threadCreate

TME Time API

Overview

The time Application Programming Interface (API) is used primarily for getting the current time in various formats. The most commonly used time function is `mpeos_timeGetMillis()`, which is used by the Java Virtual Machine (JVM) to acquire the current time expressed in milliseconds since midnight January 1, 1970. The other time-based functions are provided for convenience within the native layer.

Additionally, the JVM can make use of support for the CPU time used by particular threads. If the target operating system does not support acquiring the CPU utilization time of a thread, then 0 should be returned by the calling function.

Before reading this chapter, you should be:

- ◆ familiar with the semantics of standard C library support for date and time values

After reading this chapter, you should be able to port the time functionality within the OCAP stack

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the time functions:

`MPE_EINVAL`

`MPE_SUCCESS`

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPE_EINVAL`

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

`MPE_SUCCESS`

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h` and `mpeos_time.h`, are used by the time functions:

| | |
|----------------------------|-----------------------------|
| <code>mpe_TimeClock</code> | <code>mpe_Error</code> |
| <code>mpe_Time</code> | <code>mpe_TimeMillis</code> |
| <code>mpe_TimeVal</code> | <code>mpe_TimeTm</code> |

`mpe_TimeClock`

`mpe_TimeClock` is a target-specific value usually representing a number of system clock ticks. `mpe_TimeClock` is defined in `mpeos_time.h` as follows:

```
typedef os_Clock mpe_TimeClock;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_Time`

`mpe_Time` specifies the time in seconds. `mpe_Time` is defined in `mpeos_time.h` as follows:

```
typedef os_Time mpe_Time;
```

`mpe_TimeMillis`

`mpe_TimeMillis` specifies the time in milliseconds. `mpe_TimeMillis` is defined in `mpeos_time.h` as follows:

```
typedef os_TimeMillis mpe_TimeMillis;
```

`mpe_TimeVal`

`mpe_TimeVal` is a time representation value commonly used in supporting the BSD style MPEOS Socket API. `mpe_TimeVal` is defined in `mpeos_time.h` as follows:

```
typedef os_TimeVal mpe_TimeVal;
```

`mpe_TimeTm`

`mpe_TimeTm` specifies the time as a structure of time components (for example, year, month, day, hour, minutes, seconds, etc.). This is usually analogous for the struct `tm` value commonly found in standard C-library implementations. `mpe_TimeTm` is defined in `mpeos_time.h` as follows:

```
typedef os_TimeTm mpe_TimeTm;
```

Supported functions

The following functions need to be supported:

`mpeos_timeClock`

gets the best approximation of the processor time used by process/thread.

`mpeos_timeClockTicks`

gets the number of clock ticks per second.

`mpeos_timeClockToMillis`

converts clock time to milliseconds.

`mpeos_timeClockToTime`

converts an `mpe_TimeClock` value to an `mpe_Time` value.

`mpeos_timeGet`

gets the current time.

`mpeos_timeGetMillis`

gets the current time in milliseconds.

`mpeos_timeMillisToClock`

converts a time value to system clock ticks.

`mpeos_timeSystemClock`

gets the current value of the system clock ticks.

`mpeos_timeTimeToClock`

converts an `mpe_Time` value to a system clock tick value.

`mpeos_timeTmToTime`

converts an `mpe_TimeTm` structure to an `mpe_Time` value.

`mpeos_timeToDate`

converts the time value to the local date value.

mpeos_timeClock

gets the best approximation of the processor time used by process/thread.

syntax `mpe_TimeClock mpeos_timeClock(void);`

parameter(s) none

value returned If the call is successful, `mpeos_timeClock()` should return the amount of CPU time used by the current process or thread. The time is expressed in clock ticks. If this function is not supported by the implementation it should return 0.

description `mpeos_timeClock()` should get the best approximation of the processor time used by the current process or thread.

related function(s) `mpeos_timeClockTicks`
`mpeos_timeClockToMillis`
`mpeos_timeClockToTime`

mpeos_timeClockTicks

gets the number of clock ticks per second.

syntax mpe_TimeClock mpeos_timeClockTicks(void);

parameter(s) none

value returned If the call is successful, mpeos_timeClockTicks() should return the number of system clock ticks per second. If this function is not supported by the implementation, it should return 0.

description mpeos_timeClockTicks() should get the number of clock ticks per second used in the system for time keeping.

related function(s) mpeos_timeClock
mpeos_timeClockToMillis
mpeos_timeClockToTime

mpeos_timeClockToMillis

converts clock time to milliseconds.

syntax `uint32_t mpeos_timeClockToMillis(mpe_TimeClock clock);`

parameter(s) `clock` specifies the number of clock ticks to convert. `clock` is returned by a previous call to `mpeos_timeClock()`, `mpeos_timeClockTicks`, `mpeos_timeMillisToClock()`, or `mpeos_timeSystemClock()`. `mpe_TimeClock` is described in the `mpe_TimeClock` section.

value returned If the call is successful, `mpeos_timeClockToMillis()` should return the number of milliseconds represented by `clock`. If this function is not supported by the implementation, it should return 0.

description `mpeos_timeClockToMillis()` converts the `mpe_TimeClock` time specified by `clock` to a `uint32_t` millisecond value.

related function(s) `mpeos_timeClock`
`mpeos_timeClockTicks`
`mpeos_timeClockToTime`

mpeos_timeClockToTime

converts an mpe_TimeClock value to an mpe_Time value.

syntax mpe_Time mpeos_timeClockToTime(mpe_TimeClock *clock*);

parameter(s) *clock* specifies the number of clock ticks to convert. *clock* is returned by a previous call to mpeos_timeClock(), mpeos_timeClockTicks, mpeos_timeMillisToClock(), or mpeos_timeSystemClock(). mpe_TimeClock is described in the *mpe_TimeClock* section.

value returned If the call is successful, mpeos_timeClockToTime() should return the converted time value. If this function is not supported by the implementation, it should return 0.

description mpeos_timeClockToTime() should convert an mpe_TimeClock value to an mpe_Time value.

related function(s) mpeos_timeClock
mpeos_timeClockTicks
mpeos_timeClockToMillis
mpeos_timeSystemClock

mpeos_timeGet

gets the current time.

syntax mpe_Error mpeos_timeGet(mpe_Time * *time*);

parameter(s) *time* is an output pointer for returning the current time. *mpe_Time* is described in the *mpe_Time* section.

value returned If the call is successful, *mpeos_timeGet()* should return *MPE_SUCCESS*. Otherwise, it should return the following error code:

MPE_EINVAL indicates *time* has an invalid value.

description *mpeos_timeGet()* should get the current time. The current time is expressed as the number of seconds since 12:00 am on January 1, 1970, Greenwich Mean Time (GMT).

related function(s) *mpeos_timeGetMillis*

mpeos_timeGetMillis

gets the current time in milliseconds.

syntax mpe_Error mpeos_timeGetMillis(mpe_TimeMillis * *time*);

parameter(s) *time* is an output pointer for returning the current time in milliseconds. *mpe_TimeMillis* is described in the *mpe_TimeMillis* section.

value returned If the call is successful, *mpeos_timeGetMillis()* should return MPE_SUCCESS. Otherwise, it should return the following error code:
MPE EINVAL indicates *time* has an invalid value.

description *mpeos_timeGetMillis()* should get the current time in milliseconds. The current time is expressed as the number of seconds since 12:00 am on January 1, 1970, Greenwich Mean Time (GMT).

related function(s) *mpeos_timeGet*

mpeos_timeMillisToClock

converts a time value to system clock ticks.

syntax `mpe_TimeClock mpeos_timeMillisToClock(uint32_t milliseconds);`

parameter(s) `milliseconds` specifies the number of milliseconds to convert to clock cycles.

value returned If the call is successful, `mpeos_timeMillisToClock()` should return the converted value expressed as ticks. If this function is not supported by the implementation, it should return 0.

description `mpeos_timeMillisToClock()` should convert the millisecond time period specified by `milliseconds` to clock cycles.

related function(s) `mpeos_timeClockToMillis`
`mpeos_timeGetMillis`

mpeos_timeSystemClock

gets the current value of the system clock ticks.

syntax `mpe_TimeClock mpeos_timeSystemClock(void);`

parameter(s) none

value returned If the call is successful, `mpeos_timeSystemClock()` should return the converted value expressed as ticks. If this function is not supported by the implementation, it should return 0.

description `mpeos_timeSystemClock()` should get the current value of the system clock cycles.

related function(s) `mpeos_timeClock`
`mpeos_timeClockTicks`

mpeos_timeTimeToClock

converts an mpe_Time value to a system clock tick value.

syntax mpe_TimeClock mpeos_timeTimeToClock(mpe_Time *time*);

parameter(s) *time* specifies the time value in seconds to convert to clock ticks.
time is returned by a previous call to mpeos_timeGet() or
mpeos_timeClockToTime(). mpe_Time is described in the
mpe_Time section.

value returned If the call is successful, mpeos_timeTimeToClock() should return the converted value expressed in clock ticks. If this function is not supported by the implementation, it should return 0.

description mpeos_timeTimeToClock() should convert the mpe_Time value specified by *time* to an mpe_TimeClock value.

related function(s) mpeos_timeGet
mpeos_timeClockToTime

mpeos_timeTmToTime

converts an mpe_TimeTm structure to an mpe_Time value.

syntax mpe_Time mpeos_timeTmToTime(mpe_TimeTm * tm);

parameter(s) *tm* is an input pointer to the time component structure containing the time value to convert. *tm* is returned by a prior call to mpeos_timeToDate(). mpe_TimeTm is described in the *mpe_TimeTm* section.

value returned If the call is successful, mpeos_timeTmToTime() should return the converted value in seconds since 12:00 am January 1, 1970, Greenwich Mean Time (GMT). If this function is not supported by the implementation, it should return 0.

description mpeos_timeTmToTime() should convert the mpe_TimeTm structure specified by *tm* to an mpe_Time value.

related function(s) mpeos_timeClockToTime
mpeos_timeTimeToClock

mpeos_timeToDate

converts the time value to the local date value.

syntax mpe_Error mpeos_timeToDate(
 mpe_Time *time*,
 mpe_TimeTm * *tm*);

parameter(s) *time* specifies the time value to convert. *time* is returned by a previous call to `mpeos_timeGet()` or `mpeos_timeClockToTime()`. `mpe_Time` is described in the `mpe_Time` section.

tm is an output pointer to the time structure to be populated with the local date information. `mpe_TimeTm` is described in the `mpe_TimeTm` section.

value returned If the call is successful, `mpeos_timeToDate()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE EINVAL` indicates *time* has an invalid value.

description `mpeos_timeToDate()` should convert the time value specified by *time* to the local date value stored in *tm*.

related function(s) none

UTL Utility API

Overview

The utility Application Programming Interface (API) provides the routine utility functions to upper software layers in the OCAP stack.

The Multimedia Platform Environment Operating System (MPEOS) environment variable support is useful for various forms of static and dynamic configuration information. The `setjmp` and `longjmp` operations can be useful for error recovery conditions. The status and information APIs, similar to the environment variables, are useful for runtime status (for example, power-level status) and configuration of the host platform (for example, the number of tuners present).

Before reading this chapter, you should be familiar with:

- ◆ the semantics of standard C library support for environment variables
- ◆ the semantics of standard C library support for `setjmp` and `longjmp`
- ◆ the OpenCable Application Platform (OCAP) power and set-top box reset requirements
- ◆ the use of the Multimedia Platform Environment (MPE) initialization manager to provide environment configuration parameters

After reading this chapter, you should be able to port the utility functionality within the OCAP stack

Definitions

The following definitions, which are defined in `mpeos_util.h`, are used by the utility functions:

| | |
|--------------------------------|-------------------------------|
| <code>MPE_BS_MPE_LOWLVL</code> | <code>MPE_BS_MPE_MGRS</code> |
| <code>MPE_BS_NET_1WAY</code> | <code>MPE_BS_NET_2WAY</code> |
| <code>MPE_BS_NET_MASK</code> | <code>MPE_BS_NET_NOWAY</code> |

NOTE: The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

`MPE_BS_MPE_LOWLVL`

`MPE_BS_MPE_LOWLVL` indicates low-level initialization is done.

`MPE_BS_MPE_LOWLVL` is defined as follows:

```
#define MPE_BS_MPE_LOWLVL 0x00000001
```

`MPE_BS_MPE_MGRS`

`MPE_BS_MPE_MGRS` indicates MPE managers are installed. `MPE_BS_MPE_MGRS` is defined as follows:

```
#define MPE_BS_MPE_MGRS 0x00000002
```

`MPE_BS_NET_1WAY`

`MPE_BS_NET_1WAY` indicates broadcast and Forward Data Channel (FDC) modes are available. Reverse Data Channel (RDC) is not available.

`MPE_BS_NET_1WAY` is defined as follows:

```
#define MPE_BS_NET_1WAY 0x00000020
```

`MPE_BS_NET_2WAY`

`MPE_BS_NET_2WAY` indicates broadcast, FDC, and RDC modes are available. `MPE_BS_NET_2WAY` is defined as follows:

```
#define MPE_BS_NET_2WAY 0x00000040
```

`MPE_BS_NET_MASK`

`MPE_BS_NET_MASK` indicates the mask for network status bits.

`MPE_BS_NET_MASK` is defined as follows:

```
#define MPE_BS_NET_MASK 0x00000070
```

`MPE_BS_NET_NOWAY`

`MPE_BS_NET_NOWAY` indicates FDC and RDC modes are not available.

`MPE_BS_NET_NOWAY` is defined as follows:

```
#define MPE_BS_NET_NOWAY 0x00000010
```

Error codes

The following error codes, which are defined in `mpe_error.h`, are used by the utility functions:

MPE_EINVAL
MPE_SUCCESS

MPE_ENOMEM

- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.

MPE_EINVAL

MPE_EINVAL indicates at least one input parameter to the function has an invalid value. MPE_EINVAL is defined as follows:

```
#define MPE_EINVAL OS_EINVAL
```

MPE_ENOMEM

MPE_ENOMEM indicates the function failed due to insufficient memory resource. MPE_ENOMEM is defined as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

MPE_SUCCESS

MPE_SUCCESS indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). MPE_SUCCESS is defined as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

Data types and structures

The following data types and structures, which are defined in `mpe_types.h`, `mpeos_event.h`, and `mpeos_util.h`, are used by the utility functions:

| | |
|------------------------------|------------------------------|
| <code>mpe_Bool</code> | <code>mpe_Error</code> |
| <code>mpe_EventQueue</code> | <code>mpe_JmpBuf</code> |
| <code>mpe_PowerStatus</code> | <code>mpe_STBBootMode</code> |

`mpe_Bool`

`mpe_Bool` specifies whether the option is TRUE or FALSE. `mpe_Bool` is defined in `mpe_types.h` as follows:

```
typedef int mpe_Bool;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

`mpe_JmpBuf`

`mpe_JmpBuf` specifies the setjmp/longjmp implementation buffer. `mpe_JmpBuf` is defined in `mpeos_util.h` as follows:

```
typedef os_JmpBuf mpe_JmpBuf;
```

`mpe_STBBootMode`

`mpe_STBBootMode` specifies the mode of the set-top box bootstrap. `mpe_STBBootMode` is defined in `mpeos_util.h` as follows:

```
typedef enum {
    MPE_BOOTMODE_RESET =1,
    MPE_BOOTMODE_FAST
} mpe_STBBootMode;
```

where

`MPE_BOOTMODE_RESET`

resets the client's set-top box and is equivalent to power-on reset.

`MPE_BOOTMODE_FAST`

attempts immediate two-way (FDC+RDC) connection.

Events

The following utility events, which are defined in `mpe_types.h`, are used by the utility functions:

`mpe_PowerStatus`

`mpe_PowerStatus` is used by `mpeos_registerForPowerKey()` to indicate a set-top box power status change to all registered listeners.

`mpe_PowerStatus` is defined in `mpe_types.h` as follows:

```
typedef enum {
    MPE_POWER_FULL = 1,
    MPE_POWER_STANDBY,
} mpe_PowerStatus;
```

where

`MPE_POWER_FULL`

specifies the full power mode.

`MPE_POWER_STANDBY`

specifies the standby (low) power mode.

Supported functions

The following utility functions need to be supported:

`mpeos_envGet` gets the value of the specified environment variable.

`mpeos_envInit` initializes the environment variables.

`mpeos_envSet` sets the value of the specified environment variable.

`mpeos_longJmp` performs a non-local dispatch to the saved OCAP-stack context.

`mpeos_registerForPowerKey`
registers a handle and queue of power mode changes.

`mpeos_setJmp` saves the current OCAP-stack context for subsequent non-local dispatch.

`mpeos_stbBoot` performs bootstrap related operations.

`mpeos_stbBootStatus`
reports the boot status of the OCAP stack with or without updating.

`mpeos_stbGetAcOutletState`
gets the current AC outlet state.

`mpeos_stbGetPowerStatus`
gets the current set-top box power status.

`mpeos_stbGetRootCerts`
gets the initial set of root certificates for the platform.

`mpeos_stbIsHdCapable`
checks whether the set-top box is high-definition capable.

`mpeos_stbSetAcOutletState`
sets the current AC outlet state.

mpeos_envGet

gets the value of the specified environment variable.

syntax `const char * mpeos_envGet(const char * name);`

parameter(s) `name` is an input pointer to the name of the target environment variable.

value returned If the call is successful, `mpeos_envGet()` should return a pointer to the associated string value of the target environment variable. Otherwise, it should return `NULL`.

description `mpeos_envGet()` should get the value of the specified environment variable from the target `mpeenv.ini` file.

-
- ◆ **NOTE:** At restart, environment variables are reset to the original values as defined in the `mpeenv.ini` file.
-

related function(s) `mpeos_envSet`

mpeos_envInit

initializes the environment variables.

syntax void mpeos_envInit(void);

parameter(s) none

value returned none

description mpeos_envInit() should initialize the environment variables before using the environment. It should attempt to load the host device environment variable file. If that fails, it should load the mpeenv.ini file. Values are accessible with a call to mpeos_envGet().

related function(s) mpeos_envGet
mpeos_envSet

mpeos_envSet

sets the value of the specified environment variable.

syntax mpe_Error mpeos_envSet(
 char * name,
 char * value);

parameter(s) *name* is an input pointer to the name of the target environment variable.

value is a input pointer to a NULL-terminated value string.

value returned If the call is successful, mpeos_envSet() should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

 MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

 MPE_ENOMEM indicates the function failed due to insufficient memory resources.

description mpeos_envSet() should set the value of the specified environment variable in the host device environment variable file. mpeos_envSet() does not modify the mpeenv.ini.

- ◆ **NOTE:** At restart, environment variables are reset to the original values as defined in the mpeenv.ini file.

related function(s) mpeos_envGet

mpeos_longJmp

performs a non-local dispatch to the saved OCAP-stack context.

syntax void mpeos_longJmp(
 mpe_JmpBuf *jmpBuf*,
 int *value*);

parameter(s) *jmpBuf* specifies the jump buffer context to restore. *mpe_JmpBuf* is described in the *mpe_JmpBuf* section.

value specifies the return value for the associated *mpeos_setJmp()* operation.

value returned none

description *mpeos_longJmp()* should perform a non-local dispatch (jump) to a saved OCAP-stack context and return a value for the associated *mpeos_setJmp()* operation.

related function(s) *mpeos_setJmp*

mpeos_registerForPowerKey

registers a handle and queue of power mode changes.

syntax `mpe_Error mpeos_registerForPowerKey(mpe_EventQueue queueId , void * act);`

parameter(s) `queueId` identifies the queue in which to send the events associated with power transitions. `mpe_EventQueue` is described in the `mpe_EventQueue` section. Currently, the following events are supported:

`MPE_POWER_FULL`
specifies the full-power mode.

`MPE_POWER_STANDBY`
specifies the standby (low) power mode.

`act` is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous events are sent to the queue specified in `queueId` (via `mpeos_eventQueueSend()`), the event should pass this `act` pointer in the *optionalEventData2* field.

value returned If the call is successful, `mpeos_registerForPowerKey()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_registerForPowerKey()` should register an event queue in which to send power transition events. When a power transition is recognized, `mpeos_registerForPowerKey()` should send either an `MPE_POWER_STANDBY` or `MPE_POWER_FULL` power event (via `mpeos_eventQueueSend()`) to the `act` value passed as the *optionalEventData2* parameter (zeros should be passed for *optionalEventData1* and *optionalEventData3*).

related function(s) none

mpeos_setJmp

saves the current OCAP-stack context for subsequent non-local dispatch.

syntax `int mpeos_setJmp(mpe_JmpBuf jmpBuf);`

parameter(s) `jmpBuf` specifies the jump buffer for saving the context information. `mpe_JmpBuf` is described in the `mpe_JmpBuf` section.

value returned If the call is successful, an integer value should indicate the return context. A value of 0 should indicate a return from a direct call. A value of non-zero should indicate an indirect return from a `mpeos_longJmp()` operation that restored the saved OCAP-stack image contents.

description `mpeos_setJmp()` should save the current OCAP-stack context for subsequent non-local dispatch (`jump`).

related function(s) `mpeos_longJmp`

mpeos_stbBoot

performs bootstrap related operations.

syntax mpe_Error mpeos_stbBoot(mpe_STBBootMode mode);

parameter(s) *mode* specifies either the reset or two-way connection mode. *mpe_STBBootMode* is described in the *mpe_STBBootMode* section. Currently, the following values are supported for *mode*:

MPE_BOOTMODE_RESET

resets the client's set-top box and is equivalent to power-on reset.

MPE_BOOTMODE_FAST

attempts an immediate two-way (FDC+RDC) connection.

value returned If the call is successful, *mpeos_stbBoot()* should reboot the set-top box and return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *mode* has an invalid value.

description *mpeos_stbBoot()* should reboot the entire set-top box enabling forward and reverse data channel related function(s).

related function(s) mpeos_envGet
mpeos_stbBoot

mpeos_stbBootStatus

reports the boot status of the OCAP stack with or without updating.

syntax `uint32_t mpeos_stbBootStatus(
 mpe_Bool update,
 uint32_t statusBits,
 uint32_t bitMask);`

| | | |
|---------------------|--------------------------------|---|
| parameter(s) | <i>update</i> | determines whether or not to update the boot status. If <i>update</i> is TRUE, the boot status is updated and the current status is returned. If <i>update</i> is FALSE, the current boot status is returned without updating. <i>mpe_Bool</i> is described in the <i>mpe_Bool</i> section. |
| | <i>statusBits</i> | specifies a bit pattern for setting status bits. The bits are logically ORed with the current boot status. Currently, the following values are supported for <i>statusBits</i> : |
| | <code>MPE_BS_MPE_LOWLVL</code> | indicates low-level initialization is done. |
| | <code>MPE_BS_MPE_MGRS</code> | indicates MPE managers are installed. |
| | <code>MPE_BS_NET_1WAY</code> | indicates broadcast and FDC modes are available. RDC is not available. |
| | <code>MPE_BS_NET_2WAY</code> | indicates broadcast, FDC, and RDC are available. |
| | <code>MPE_BS_NET_MASK</code> | indicates the mask for network status bits. |
| | <code>MPE_BS_NET_NOWAY</code> | indicates FDC and RDC modes are not available. |
| | <i>bitMask</i> | specifies a bit pattern for clearing status bits. If no bits are to be cleared, <code>0xFFFFFFFF</code> should be passed. |

value returned If the call is successful, `mpeos_stbBootStatus()` should return a 32-bit pattern indicating the status of the various elements of the system.

description `mpeos_stbBootStatus()` should update and report, or should report without updating, the boot status of the OCAP stack. The boot status is a 32-bit value with each bit defined to represent a stage of the boot process or the availability of a resource (for example, network status).

If *update* is TRUE, the call is an update call and *statusBits* contains the status bits to be set. *bitMask* may be used to clear any status bits. If *update* is FALSE, the call is a polling call and *statusBits* and *bitMask* are ignored.

-
- ◆ **NOTE:** Due to the dynamic nature of some status bits (for example, network status), an implementation may need to actively acquire portions of the current status.
-

related function(s) mpeos_envGet
mpeos_stbBoot

mpeos_stbGetAcOutletState

gets the current AC outlet state.

syntax mpe_Error mpeos_stbGetAcOutletState(mpe_Bool * state);

parameter(s) *state* is an output pointer to the current state of the external AC outlet. If the returned value is TRUE the outlet is enabled. If the return value is FALSE the outlet is disabled.

value returned If the call is successful, mpeos_stbGetAcOutletState() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates *state* has an invalid value.

description mpeos_stbGetAcOutletState() should get the current AC outlet state.

related function(s) mpeos_stbSetAcOutletState

mpeos_stbGetPowerStatus

gets the current set-top box power status.

syntax mpe_PowerStatus mpeos_stbGetPowerStatus(void);

parameter(s) none

value returned If the power is on, mpeos_stbGetPowerStatus() should return MPE_POWER_FULL. Otherwise, it should return MPE_POWER_STANDBY.

description mpeos_stbGetPowerStatus() should get the current status of the set-top box power.

related function(s) mpeos_stbSetAcOutletState

mpeos_stbGetRootCerts

gets the initial set of root certificates for the platform.

syntax mpe_Error mpeos_stbGetRootCerts(
 uint8_t ** roots,
 uint32_t * len);

parameter(s) *roots* is an output pointer for returning a pointer to the memory location containing the platform's root certificate(s).

len is an output pointer for returning the size in bytes of the memory location containing the roots.

value returned If the call is successful, mpeos_stbGetRootCerts() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_EINVAL indicates at least one parameter to the function has an invalid value.

description mpeos_stbGetRootCerts() should get the initial set of root certificates for the platform.

related function(s) none

mpeos_stbIsHdCapable

checks whether the set-top box is high-definition capable.

syntax mpe_Bool mpeos_stbIsHdCapable(void);

parameter(s) none

value returned If the set-top box is high-definition capable, mpeos_stbIsHdCapable() should return TRUE. Otherwise, it should return FALSE.

description mpeos_stbIsHdCapable() should check whether the set-top box is high-definition capable or not.

related function(s) none

mpeos_stbSetAcOutletState

sets the current AC outlet state.

syntax `mpe_Error mpeos_stbSetAcOutletState(mpe_Bool state);`

parameter(s) `state` determines the new state of the AC outlet. If `state` is set to TRUE, the AC outlet is enabled. If `state` is set to FALSE, the AC outlet is disabled. `mpe_Bool` is described in the *mpe_Bool* section.

value returned If the call is successful, `mpeos_stbSetAcOutletState()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:
`MPE_EINVAL` indicates `state` has an invalid value.

description `mpeos_stbSetAcOutletState()` should set the new AC outlet state.

related function(s) `mpeos_stbGetAcOutletState`

VBI Vertical Blanking Interval API

Overview

The Vertical Blanking Interval (VBI) Application Programming Interface (API) provides a way to extract VBI data from analog scan lines or MPEG digital video. Closed captioning is one of the things carried via VBI.

Particular VBI data formats must be supported by the MPEOS VBI API. Specifically, the implementation must support retrieval of CC1, CC2, T1, and T2 from field 1 of VBI line 21 from analog video sources; and CC3, CC4, T3, T4, and XDS from field 2 of VBI line 21 from analog video sources. Other data sources and formats, including digital, are supportable via the API, but are not required for the port.

Before reading this chapter, you should be familiar with:

- ◆ the VBI data formats as defined by EIA/CEA-608-B (LINE 21 DATA SERVICES)
- ◆ the `org.ocap.media.VBIFilter` and `org.ocap.media.VBIFilterGroup` as defined by the OCAP Specification

After reading this chapter, you should be:

- ◆ familiar with the basic operation of VBI data retrieval and filtering
- ◆ familiar with the required and optional VBI functionality relevant to an MPEOS port
- ◆ able to port the VBI functionality within the OCAP stack

Definitions

The following VBI definitions, which are defined in `mpeos_vbi.h`, are used by the VBI functions:

```
MPE_VBI_OPTION_CLEAR_BUFFER  
MPE_VBI_OPTION_CLEAR_READ_DATA  MPE_VBI_OPTION_IF_NOT_STOPPED
```

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definitions as the values may change in a future release.
-

MPE_VBI_OPTION_CLEAR_BUFFER

`MPE_VBI_OPTION_CLEAR_BUFFER` indicates that upon successful completion the associated operation should clear the associated data buffer of all contents. `MPE_VBI_OPTION_CLEAR_BUFFER` is defined as follows:

```
#define MPE_VBI_OPTION_CLEAR_BUFFER (0x00000002)
```

MPE_VBI_OPTION_CLEAR_READ_DATA

`MPE_VBI_OPTION_CLEAR_READ_DATA` indicates that upon successful completion the associated operation should clear only the read data from the data buffer. `MPE_VBI_OPTION_CLEAR_READ_DATA` is defined as follows:

```
#define MPE_VBI_OPTION_CLEAR_READ_DATA (0x00000004)
```

MPE_VBI_OPTION_IF_NOT_STOPPED

`MPE_VBI_OPTION_IF_NOT_STOPPED` indicates the associated operation will be carried out only if the filter session is not in the stopped state.

`MPE_VBI_OPTION_IF_NOT_STOPPED` is defined as follows:

```
#define MPE_VBI_OPTION_IF_NOT_STOPPED (0x00000001)
```

Error codes

The following error codes, which are defined in `mpe_error.h` and `mpeos_vbi.h`, are used by the VBI functions:

| | |
|---|--|
| <code>MPE_EINVAL</code> | <code>MPE_VBI_ERROR</code> |
| <code>MPE_VBI_ERROR_FILTER_NOT_AVAILABLE</code> | |
| <code>MPE_SUCCESS</code> | |
| <code>MPE_VBI_ERROR_INVALID_FILTER_SESSION</code> | |
| <code>MPE_VBI_ERROR_SOURCE_CLOSED</code> | |
| <code>MPE_VBI_ERROR_SOURCE_SCRAMBLED</code> | <code>MPE_VBI_ERROR_UNSUPPORTED</code> |

-
- ◆ **NOTE:** The actual definitions are included here for informational purposes only. You should not need to change the value of the definitions, nor should you use the values listed instead of the definition as the values may change in a future release.
-

`MPE_EINVAL`

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value. `MPE_EINVAL` is defined in `mpe_error.h` as follows:

```
#define MPE_EINVAL OS_EINVAL
```

`MPE_ENOMEM`

`MPE_ENOMEM` indicates the function failed due to insufficient memory resources. `MPE_ENOMEM` is defined in `mpe_error.h` as follows:

```
#define MPE_ENOMEM OS_ENOMEM
```

`MPE_SUCCESS`

`MPE_SUCCESS` indicates successful completion of an `mpe_` or `mpeos_` function call. This is the value that should be compared against to determine the function completion status. Implementation code should not assume any specific value (for example, zero). `MPE_SUCCESS` is defined in `mpe_error.h` as follows:

```
#define MPE_SUCCESS OS_SUCCESS
```

`MPE_VBI_ERROR`

`MPE_VBI_ERROR` indicates initialization failed. `MPE_VBI_ERROR` is defined in `mpeos_vbi.h` as follows:

```
#define MPE_VBI_ERROR (0x00002200)
```

`MPE_VBI_ERROR_FILTER_NOT_AVAILABLE`

`MPE_VBI_ERROR_FILTER_NOT_AVAILABLE` indicates insufficient filtering resources for the requested filter specification.

`MPE_VBI_ERROR_FILTER_NOT_AVAILABLE` is defined in `mpeos_vbi.h` as follows:

```
#define MPE_VBI_ERROR_FILTER_NOT_AVAILABLE (MPE_VBI_ERROR + 1)
```

MPE_VBI_ERROR_INVALID_FILTER_SESSION

MPE_VBI_ERROR_INVALID_FILTER_SESSION indicates filterSession is invalid. MPE_VBI_ERROR_INVALID_FILTER_SESSION is defined in `mpeos_vbi.h` as follows:

```
#define MPE_VBI_ERROR_INVALID_FILTER_SESSION  
    (MPE_VBI_ERROR + 2)
```

MPE_VBI_ERROR_SOURCE_CLOSED

MPE_VBI_ERROR_SOURCE_CLOSED indicates the specified filter source is no longer active. MPE_VBI_ERROR_SOURCE_CLOSED is defined in `mpeos_vbi.h` as follows:

```
#define MPE_VBI_ERROR_SOURCE_CLOSED (MPE_VBI_ERROR + 3)
```

MPE_VBI_ERROR_SOURCE_SCRAMBLED

MPE_VBI_ERROR_SOURCE_SCRAMBLED indicates the specified filter source cannot be unscrambled. MPE_VBI_ERROR_SOURCE_SCRAMBLED is defined in `mpeos_vbi.h` as follows:

```
#define MPE_VBI_ERROR_SOURCE_SCRAMBLED (MPE_VBI_ERROR + 4)
```

MPE_VBI_ERROR_UNSUPPORTED

MPE_VBI_ERROR_UNSUPPORTED indicates the specified filter source is unsupported. MPE_VBI_ERROR_UNSUPPORTED is defined in `mpeos_vbi.h` as follows:

```
#define MPE_VBI_ERROR_UNSUPPORTED (MPE_VBI_ERROR + 5)
```

Data types and structures

The following data types and structures, which are defined in `mpe_error.h`, `mpeos_dvr.h`, `mpeos_event.h`, `mpeos_filter.h`, `mpeos_media.h`, and `mpeos_vbi.h`, are used by the VBI functions:

| | |
|--------------------------------------|-------------------------------------|
| <code>mpe_DvrPlayback</code> | <code>mpe_Error</code> |
| <code>mpe_EventQueue</code> | <code>mpe_FilterComponent</code> |
| <code>mpe_FilterSpec</code> | <code>mpe_MediaDecodeSession</code> |
| <code>mpe_VBIDataFormat</code> | <code>mpe_VBIFields</code> |
| <code>mpe_VBIFilterSession</code> | <code>mpe_VBIParameter</code> |
| <code>mpe_VBISessionParameter</code> | <code>mpe_VBISource</code> |
| <code>mpe_VBISourceParams</code> | <code>mpe_VBISourceType</code> |

`mpe_DvrPlayback`

`mpe_DvrPlayback` is an opaque handle for the native playback data structure. The handle is used to control the rate and position of the playback. `mpe_DvrPlayback` is described in `mpeos_dvr.h` as follows:

```
typedef struct _mpe_DvrPlaybackH { int unused1; }
* mpe_DvrPlayback;
```

`mpe_Error`

`mpe_Error` specifies the type of error codes. The MPE error codes are defined in `mpe_error.h`. `mpe_Error` is defined in `mpe_types.h` as follows:

```
typedef uint32_t mpe_Error;
```

`mpe_EventQueue`

`mpe_EventQueue` defines the event queue type. `mpe_EventQueue` is defined in `mpeos_event.h` as follows:

```
typedef os_EventQueue mpe_EventQueue;
```

`mpe_FilterComponent`

`mpe_FilterComponent` specifies a set of data values which can be compared against a target data array. `mpe_FilterComponent` is defined in `mpeos_filter.h` as follows:

```
typedef struct mpe_FilterComponent {
    uint32_t length;
    uint8_t * mask;
    uint8_t * values;
} mpe_FilterComponent;
```

where

`length` specifies the length, in bytes, of the mask and value arrays.

`mask` is an input pointer to an array of bytes defining the mask to be applied to the target data. The array of bytes needs to be `length` in bytes, at least.

`values` is an input pointer to an array of bytes defining the values to be compared against in the target data. The array of bytes needs to be `length` in bytes, at least.

mpe_FilterSpec

`mpe_FilterSpec` specifies the terms that define a filter. The filter specification describes the conditional terms to be evaluated against the target MPEG table section. If the target section positive components and negative components are considered as arbitrarily-long data words, the logical expression of the filter is:

```
((sectiondata[] & pos.mask) == pos.vals)
&& !((sectiondata[] & neg.mask) == neg.vals).
```

`mpe_FilterSpec` is defined in `mpeos_filter.h` as follows:

```
typedef struct mpe_FilterSpec {
    mpe_FilterComponent pos;
    mpe_FilterComponent neg;
} mpe_FilterSpec;
```

where

- | | |
|-----|--|
| pos | specifies the terms that need to be present in the target data, for the filter described with this <code>mpe_FilterSpec</code> , to signal a match. If pos length equals 0, the filter is a negative-only filter. <code>mpe_FilterComponent</code> is described in the <i>mpe_FilterComponent</i> section. |
| neg | specifies the terms that do not need to be present in the target data for the filter described with this <code>mpe_FilterSpec</code> , to signal a match. If neg length equals 0, the filter is a positive-only filter. <code>mpe_FilterComponent</code> is described in the <i>mpe_FilterComponent</i> section. |

mpe_MediaDecodeSession

`mpe_MediaDecodeSession` defines the media decode session handle. `mpe_MediaDecodeSession` is defined in `mpeos_media.h` as follows:

```
typedef struct _mpe_MediaDecodeSessionH { int unused1; }
*mpe_MediaDecodeSession;
```

mpe_VBIDataFormat mpe_VBIDataFormat specifies the supported VBI data formats.

mpe_VBIDataFormat is defined in mpeos_vbi.h as follows:

```
typedef enum mpe_VBIDataFormat {
    MPE_VBI_FORMAT_XDS = 1,
    MPE_VBI_FORMAT_T1 = 2,
    MPE_VBI_FORMAT_T2 = 3,
    MPE_VBI_FORMAT_T3 = 4,
    MPE_VBI_FORMAT_T4 = 5,
    MPE_VBI_FORMAT_EIA608B_GENERIC = 6,
    MPE_VBI_FORMAT_NABTS = 7,
    MPE_VBI_FORMAT_AMOL_I = 8,
    MPE_VBI_FORMAT_AMOL_II = 9,
    MPE_VBI_FORMAT_UNKNOWN = 10,
    MPE_VBI_FORMAT_CC1 = 91,
    MPE_VBI_FORMAT_CC2 = 92,
    MPE_VBI_FORMAT_CC3 = 93,
    MPE_VBI_FORMAT_CC4 = 94
} mpe_VBIDataFormat;
```

where

MPE_VBI_FORMAT_XDS

designates the Extended Data Service (XDS) data format of line 21, field 2 as defined by EIA-608-B.

MPE_VBI_FORMAT_T1

designates the T1 text service format of VBI line 21, field 1 as defined by EIA-608-B.

MPE_VBI_FORMAT_T2

designates the T2 text service format of VBI line 21, field 1 as defined by EIA-608-B.

MPE_VBI_FORMAT_T3

designates the T3 text service format of VBI line 21, field 2 as defined by EIA-608-B.

MPE_VBI_FORMAT_T4

designates the T4 text service format of VBI line 21, field 2 as defined by EIA-608-B.

MPE_VBI_FORMAT_EIA608B_GENERIC

designates the EIA-608-B generic character one and character two format of line 21, field 1 or 2.

MPE_VBI_FORMAT_NABTS

designates the North American Broadcast Teletext Specification (NABTS) data format as defined by EIA-516.

MPE_VBI_FORMAT_AMOL_I

designates the Automated Measurement Of Lineups (AMOL I) data format as defined by ACN 403-1218-024.

MPE_VBI_FORMAT_AMOL_II

designates the AMOL II data format as defined by ACN 403-1218-024.

MPE_VBI_FORMAT_UNKNOWN
designates a non-standard VBI waveform.

MPE_VBI_FORMAT_CC1
designates the CC1 data of VBI line 21, field 1 as defined by EIA-608-B.

MPE_VBI_FORMAT_CC2
designates the CC2 data of VBI line 21, field 1 as defined by EIA-608-B.

MPE_VBI_FORMAT_CC3
designates the CC3 data of VBI line 21, field 2 as defined by EIA-608-B.

MPE_VBI_FORMAT_CC4
designates the CC4 data of VBI line 21, field 2 as defined by EIA-608-B.

mpe_VBIFields

`mpe_VBIFields` specifies the target fields for VBI data acquisition.
`mpe_VBIFields` is defined in `mpeos_vbi.h` as follows:

```
typedef enum mpe_VBIFields {
    MPE_VBI_FIELD_ODD=1,
    MPE_VBI_FIELD_EVEN=2,
    MPE_VBI_FIELD_MIXED=3
} mpe_VBIFields;
```

where

MPE_VBI_FIELD_ODD
designates the odd field (field 1) of a VBI line.

MPE_VBI_FIELD_EVEN
designates the even field (field 2) of a VBI line.

MPE_VBI_FIELD_MIXED
designates both the odd and even fields (fields 1 and 2) of a VBI line.

mpe_VBIFilterSession

`mpe_VBIFilterSession` specifies the VBI session type.
`mpe_VBIFilterSession` is defined in `mpeos_vbi.h` as follows:

```
typedef struct _mpe_VBIFilterSession { int unused1; }
*mpe_VBIFilterSession;
```

mpe_VBIPParameter

`mpe_VBIPParameter` specifies VBI subsystem parameter identifiers.
`mpe_VBIPParameter` is defined in `mpeos_vbi.h` as follows:

```
typedef enum mpe_VBIPParameter {
    MPE_VBI_PARAM_SCTE20LINE21_CAPABILITY = 1,
    MPE_VBI_PARAM_SCTE21LINE21_CAPABILITY,
    MPE_VBI_SEPARATED_FILTERING_CAPABILITY,
    MPE_VBI_MIXED_FILTERING_CAPABILITY
} mpe_VBIPParameter;
```

where

`MPE_VBI_PARAM_SCTE20LINE21_CAPABILITY`

indicates line 21 VBI data retrieval in MPEG picture headers
as defined in ANSI/SCTE 20.

`MPE_VBI_PARAM_SCTE21LINE21_CAPABILITY`

indicates line 21 VBI data retrieval in MPEG picture headers
as defined in ANSI/SCTE 21.

`MPE_VBI_SEPARATED_FILTERING_CAPABILITY`

indicates the separated-filtering data format is supported.

`MPE_VBI_MIXED_FILTERING_CAPABILITY`

indicates the mixed-filtering data format is supported.

mpe_VBISessionParameter

`mpe_VBISessionParameter` specifies VBI session parameter identifiers.
`mpe_VBISessionParameter` is defined in `mpeos_vbi.h` as follows:

```
typedef enum mpe_VBISessionParameter {
    MPE_VBI_PARAM_DATA_UNIT_THRESHOLD,
    MPE_VBI_PARAM_BUFFER_SIZE,
    MPE_VBI_PARAM_BUFFERED_DATA_SIZE
} mpe_VBISessionParameter;
```

where

`MPE_VBI_PARAM_DATA_UNIT_THRESHOLD`

indicates a value parameter designating the data unit
threshold for triggering the
`MPE_VBI_EVENT_DATAUNITS RECEIVED` notification. A value
of 0 disables this notification.

`MPE_VBI_PARAM_BUFFER_SIZE`

indicates the size, in bytes, of the VBI session data buffer.

`MPE_VBI_PARAM_BUFFERED_DATA_SIZE`

indicates the number of bytes currently in the VBI session
data buffer.

mpe_VBISource

`mpe_VBISource` specifies the VBI source parameters. `mpe_VBISource` is defined in `mpeos_vbi.h` as follows:

```
typedef struct mpe_VBISource {
    uint32_t * vbiLines;
    uint32_t vbiLineCount;
    mpe_VBIFields vbiFields;
    mpe_VBISourceType sourceType;
    mpe_VBISourceParams parm;
} mpe_VBISource;
```

where

`vbiLines` is a pointer to the array of VBI line(s) to acquire.

`vbiLineCount` specifies the number of elements in `vbiLines`.

`vbiFields` specifies the field(s) to acquire from the associated line(s). `mpe_VBIFields` is defined in the *mpe_VBIFields* section.

`sourceType` specifies the type of the VBI buffer source. `mpe_VBISourceType` is defined in the *mpe_VBISourceType* section.

`parm` specifies the type-specific parameters. `mpe_VBISourceParams` is defined in the *mpe_VBISourceParams* section.

mpe_VBISourceParams

`mpe_VBISourceParams` specifies a container for all source type-specific parameters. `mpe_VBISourceParams` is defined in `mpeos_vbi.h` as follows:

```
typedef union mpe_VBISourceParams {
    mpe_MediaDecodeSession mediaSessionID;
    mpe_DvrPlayback playbackSessionID;
} mpe_VBISourceParams;
```

where

`mediaSessionID`

specifies the media decode session handle. `mpe_MediaDecodeSession` is defined in the *mpe_MediaDecodeSession* section.

`playbackSessionID`

specifies an opaque handle for the native playback data structure. `mpe_DvrPlayback` is defined in the *mpe_DvrPlayback* section.

mpe_VBISourceType

mpe_VBISourceType specifies the VBI filtering source types.
mpe_VBISourceType is defined in mpeos_vbi.h as follows:

```
typedef enum mpe_VBISourceType {  
    MPE_VBI_SOURCE_DECODE_SESSION = 1,  
    MPE_VBI_SOURCE_PLAYBACK_SESSION  
} mpe_VBISourceType;
```

where

MPE_VBI_SOURCE_DECODE_SESSION
indicates the source is a live decode session.

MPE_VBI_SOURCE_PLAYBACK_SESSION
indicates the source is a DVR playback session.

Events

The following VBI events, which are defined in `mpeos_vbi.h`, are used by the VBI functions:

```
MPE_VBI_EVENT_BUFFER_FULL
MPE_VBI_EVENT_DATAUNITS_RECEIVED
MPE_VBI_EVENT_FILTER_AVAILABLE  MPE_VBI_EVENT_FILTER_STOPPED
MPE_VBI_EVENT_FIRST_DATAUNIT   MPE_VBI_EVENT_OUT_OF_MEMORY
MPE_VBI_EVENT_SOURCE_CLOSED
MPE_VBI_EVENT_SOURCE_SCRAMBLED MPE_VBI_EVENT_UNKNOWN
```

MPE_VBI_EVENT_BUFFER_FULL

`MPE_VBI_EVENT_BUFFER_FULL` is sent by an active VBI filter to the associated filter event queue when at least `bufferSize` (`MPE_VBI_PARAM_DATABUFFER_SIZE`) bytes are contained in the VBI session's buffer. The filtering session is stopped. `optionalEventData1` contains the originating `mpe_VBIFilterSession` and `optionalEventData2` contains the corresponding `act`. `MPE_VBI_EVENT_BUFFER_FULL` is described as follows:

```
#define MPE_VBI_EVENT_BUFFER_FULL (MPE_VBI_EVENT_UNKNOWN + 3)
```

MPE_VBI_EVENT_DATAUNITS_RECEIVED

`MPE_VBI_EVENT_DATAUNITS_RECEIVED` is sent by an active VBI filter to the associated filter event queue when at least `dataUnitNotificationThreshold` (`MPE_VBI_PARAM_DATA_UNIT_THRESHOLD`) data units are contained in the VBI session's buffer. `optionalEventData1` contains the originating `mpe_VBIFilterSession` and `optionalEventData2` contains the corresponding `act`. `MPE_VBI_EVENT_DATAUNITS_RECEIVED` is described as follows:

```
#define MPE_VBI_EVENT_DATAUNITS_RECEIVED
(MPE_VBI_EVENT_UNKNOWN + 2)
```

Data units

The definition of a data unit depends upon the particular data format as specified by `mpe_VBIDataFormat`. `mpe_VBIDataFormat` is defined in the `mpe_VBIDataFormat` section. Currently, the following values are supported for data units:

MPE_VBI_FORMAT_XDS

specifies a data unit is a single XDS packet.

MPE_VBI_FORMAT_EIA608B_GENERIC

specifies a data unit is a Character 1/2 byte pair as defined in EIA-608B.

MPE_VBI_FORMAT_T1/2/3/4

specifies a data unit is a byte sequence between a text mode in code (RTD/TR) and a text mode out (EOC) or cc/xds mode in code (RTD/TR/RCL/RDC/RU2/RU3/RU4/XDS Start).

MPE_VBI_FORMAT_CC1/2/3/4

specifies a data unit is a byte sequence between a cc mode in code (RCL/RDC/RU2/RU3/RU4) and a cc mode out (EOC) or cc/text/xds mode in code (RTD/TR/RCL/RDC/RU2/RU3/RU4/XDS Start).

MPE_VBI_FORMAT_AMOL_I/II and MPE_VBI_FORMAT_UNKNOWN

specifies a data unit is all the bits in a VBI line.

MPE_VBI_EVENT_FILTER_AVAILABLE

MPE_VBI_EVENT_FILTER_AVAILABLE is sent by an active VBI filter to the associated availability queue when memory or other resources are depleted. MPE_VBI_EVENT_FILTER_AVAILABLE is described as follows:

```
#define MPE_VBI_EVENT_FILTER_AVAILABLE  
    (MPE_VBI_EVENT_UNKNOWN + 8)
```

MPE_VBI_EVENT_FILTER_STOPPED

MPE_VBI_EVENT_FILTER_STOPPED is sent by an active VBI filter to the associated filter event queue when the filtering session has not already been put into the stopped state and `mpeos_vbiFilterStop()` is called. optionalEventData1 contains the originating `mpe_VBIFilterSession` and optionalEventData2 contains the corresponding `act`. MPE_VBI_EVENT_FILTER_STOPPED is described as follows:

```
#define MPE_VBI_EVENT_FILTER_STOPPED  
    (MPE_VBI_EVENT_UNKNOWN + 4)
```

MPE_VBI_EVENT_FIRST_DATAUNIT

MPE_VBI_EVENT_FIRST_DATAUNIT is sent by an active VBI filter to the associated filter event queue when a data unit is written to a VBI session's empty buffer. optionalEventData1 contains the originating `mpe_VBIFilterSession` and optionalEventData2 contains the corresponding `act`. MPE_VBI_EVENT_FIRST_DATAUNIT is described as follows:

```
#define MPE_VBI_EVENT_FIRST_DATAUNIT  
    (MPE_VBI_EVENT_UNKNOWN + 1)
```

MPE_VBI_EVENT_OUT_OF_MEMORY

MPE_VBI_EVENT_OUT_OF_MEMORY is sent by an active VBI filter to the associated filter event queue when memory or other resources are depleted. The filtering session is stopped. optionalEventData1 contains the originating `mpe_VBIFilterSession` and optionalEventData2 contains the corresponding `act`. MPE_VBI_EVENT_OUT_OF_MEMORY is described as follows:

```
#define MPE_VBI_EVENT_OUT_OF_MEMORY  
    (MPE_VBI_EVENT_UNKNOWN + 7)
```

MPE_VBI_EVENT_SOURCE_CLOSED

MPE_VBI_EVENT_SOURCE_CLOSED is sent by an active VBI filter to the associated filter event queue when the filtering session is stopped due to termination of the associated decode session or other resource required for VBI data acquisition. optionalEventData1 contains the originating mpe_VBIFilterSession and optionalEventData2 contains the corresponding *act*. MPE_VBI_EVENT_SOURCE_CLOSED is described as follows:

```
#define MPE_VBI_EVENT_SOURCE_CLOSED  
    (MPE_VBI_EVENT_UNKNOWN + 5)
```

MPE_VBI_EVENT_SOURCE_SCRAMBLED

MPE_VBI_EVENT_SOURCE_SCRAMBLED is sent by an active VBI filter to the associated filter event queue when the VBISource becomes scrambled, preventing extraction of VBI data. MPE_VBI_EVENT_SOURCE_SCRAMBLED does not stop the filtering session. optionalEventData1 contains the originating mpe_VBIFilterSession and optionalEventData2 contains the corresponding *act*. MPE_VBI_EVENT_SOURCE_SCRAMBLED is described as follows:

```
#define MPE_VBI_EVENT_SOURCE_SCRAMBLED  
    (MPE_VBI_EVENT_UNKNOWN + 6)
```

MPE_VBI_EVENT_UNKNOWN

MPE_VBI_EVENT_UNKNOWN indicates an unknown event occurred. MPE_VBI_EVENT_UNKNOWN is described as follows:

```
#define MPE_VBI_EVENT_UNKNOWN (0x00001200)
```

Supported functions

The following VBI functions need to be supported:

`mpeos_vbiFilterGetParam`
gets the designated parameter for a particular filtering session.

`mpeos_vbiFilterReadData`
copies data into a buffer.

`mpeos_vbiFilterRelease`
releases the filter and any associated resources.

`mpeos_vbiFilterSetParam`
sets the designated parameter on the VBI filtering session.

`mpeos_vbiFilterStart`
starts VBI data acquisition and filtering.

`mpeos_vbiFilterStop`
stops the designated filter.

`mpeos_vbiGetParam`
gets the designated parameter from the VBI filtering subsystem.

`mpeos_vbiInit` initializes the VBI filtering system.

`mpeos_vbiRegisterAvailability`
registers for available VBI filters notification.

`mpeos_vbiShutdown`
shuts down the VBI filtering system.

`mpeos_vbiUnregisterAvailability`
unregisters for available VBI filters notification.

mpeos_vbiFilterGetParam

gets the designated parameter for a particular filtering session.

syntax mpe_Error mpeos_vbiFilterGetParam(
 mpe_VBIFilterSession *filterSession*,
 mpe_VBISessionParameter *parameter*,
 uint32_t * *value*);

parameter(s) *filterSession*

specifies the filter. *filterSession* must have been returned by a previous call to `mpeos_vbiFilterStart()`.

parameter

specifies which filter parameter to return in the *value* parameter. `mpe_VBISessionParameter` is defined in the `mpe_VBISessionParameter` section. Currently, the following values are supported for *parameter*:

`MPE_VBI_PARAM_DATA_UNIT_THRESHOLD`

indicates the data unit threshold for triggering the `MPE_VBI_EVENT_DATAUNITS_RECEIVED` notification. See the discussion of

`MPE_VBI_EVENT_DATAUNITS_RECEIVED` in the *Events* section for details on which data units are in effect.

`MPE_VBI_PARAM_BUFFER_SIZE`

indicates the size, in bytes, of the VBI session data buffer.

`MPE_VBI_PARAM_BUFFERED_DATA_SIZE`

indicates the number of bytes currently in the VBI session data buffer.

value

holds the return value requested by the specified *parameter*.

value returned

If the call is successful, `mpeos_vbiFilterGetParam()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL`

indicates at least one input parameter to the function has an invalid value.

`MPE_VBI_ERROR_INVALID_FILTER_SESSION`

indicates *filterSession* is invalid.

description

`mpeos_vbiFilterGetParam()` retrieves the designated parameter for a specified VBI filtering session.

related function(s)

`mpeos_vbiFilterSetParam`

mpeos_vbiFilterReadData

copies data into a buffer.

```
syntax mpe_Error mpeos_vbiFilterReadData(
    mpe_VBIFilterSession filterSession,
    uint32_t offset,
    uint32_t byteCount,
    uint32_t flags,
    uint8_t * buffer,
    uint32_t * bytesRead );
```

parameter(s) *filterSession* specifies the filter. *filterSession* must have been returned by a previous call to `mpeos_vbiFilterStart()`. `mpe_VBIFilterSession` is defined in the `mpe_VBIFilterSession` section.

offset specifies the offset within the VBI buffer from which to read.

byteCount specifies the number of bytes to read from the VBI buffer. This can be larger than the size of the VBI buffer (for example, the target buffer size).

flags specifies extended options for reading the VBI buffer. Currently, the following values are supported by *flag*:

MPE_VBI_OPTION_CLEAR_BUFFER

indicates the entire VBI buffer associated with the VBI filter session is to be cleared upon a successful read.

MPE_VBI_OPTION_CLEAR_READ_DATA

indicates the data read from the VBI buffer is to be cleared upon a successful read. Any data bytes remaining in the buffer beyond the read data will be packed.

buffer is an output pointer to the buffer where the VBI buffer data is copied.

bytesRead is an output pointer to the destination for the number of bytes actually copied.

value returned If the call is successful, `mpeos_vbiFilterReadData()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

MPE_VBI_ERROR_INVALID_FILTER_SESSION
indicates *filterSession* is invalid.

description `mpeos_vbiFilterReadData()` should copy *byteCount*, or (VBI buffer size - *offset*) if smaller than *byteCount*, starting at *offset* bytes within the VBI buffer. The total number of bytes read should be stored in *bytesRead*, if *bytesRead* is non-null.

-
- ◆ **NOTE:** The filtering session's VBI buffer can be cleared by calling
`mpeos_vbiFilterReadData
(session,0,0,MPE_VBI_OPTION_CLEAR_BUFFER,NULL,NULL);`
-

related function(s) none

mpeos_vbiFilterRelease

releases the filter and any associated resources.

syntax mpe_Error mpeos_vbiFilterRelease(
 mpe_VBIFilterSession *filterSession*);

parameter(s) *filterSession* identifies the filter. *filterSession* must have been returned by a previous call to `mpeos_vbiFilterStart()`.
`mpe_VBIFilterSession` is defined in the `mpe_VBIFilterSession` section.

value returned If the call is successful, `mpeos_vbiFilterRelease()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_VBI_ERROR_INVALID_FILTER_SESSION`
indicates *filterSession* is invalid.

description `mpeos_vbiFilterRelease()` should release the filter and any associated resources - including the associated VBI buffer and its contents. If the filter has not already been put into the cancelled state, a `MPE_VBI_EVENT_FILTER_STOPPED` event is put in the event queue of the filtering request identified by *filterSession*.

related function(s) `mpeos_vbiFilterSetParam`

mpeos_vbiFilterSetParam

sets the designated parameter on the VBI filtering session.

syntax mpe_Error mpeos_vbiFilterSetParam(
 mpe_VBIFilterSession *filterSession*,
 mpe_VBISessionParameter *parameter*,
 uint32_t *value*);

parameter(s) *mpe_VBIFilterSession*

specifies the filter. *filterSession* must have been returned by a previous call to `mpeos_vbiFilterStart()`.

parameter

mpe_VBISessionParameter is defined in the *mpe_VBISessionParameter* section. Currently, the following values are supported for *parameter*:

MPE_VBI_PARAM_DATA_UNIT_THRESHOLD

indicates the data unit threshold for triggering the MPE_VBI_EVENT_DATAUNITS_RECEIVED notification.

value

specifies the number of data units that will trigger the MPE_VBI_EVENT_DATAUNITS_RECEIVED notification. See the discussion of MPE_VBI_EVENT_DATAUNITS_RECEIVED in the *Events* section for details on which data units are in effect.

value returned

If the call is successful, `mpeos_vbiFilterSetParam()` should return MPE_SUCCESS. Otherwise, it should return one of the following error codes:

MPE_EINVAL indicates at least one input parameter to the function has an invalid value.

MPE_VBI_ERROR_INVALID_FILTER_SESSION
 indicates *filterSession* is invalid.

description

`mpeos_vbiFilterSetParam()` should set the designated parameter on the VBI filtering session.

related function(s)

`mpeos_vbiFilterGetParam`

mpeos_vbiFilterStart

starts VBI data acquisition and filtering.

```
syntax mpe_Error mpeos_vbiFilterStart(
    mpe_VBISource * filterSource,
    mpe_FilterSpec * filterSpec,
    mpe_EventQueue queueId ,
    void * act,
    mpe_VBIDataFormat dataFormat ,
    uint32_t dataUnitSize ,
    uint32_t bufferSize ,
    uint32_t dataUnitNotificationThreshold ,
    uint32_t flags ,
    mpe_VBIFilterSession * filterSession );
```

| | | |
|---------------------|----------------------------------|--|
| parameter(s) | <i>filterSource</i> | is an input pointer to a description of the VBI data source. The filter operates on VBI data from this source. <i>mpe_VBISource</i> is described in the <i>mpe_VBISource</i> section. |
| | <i>filterSpec</i> | is an input pointer to the filter specification. This defines the criteria the filter evaluates data units against before placing them in the associated data buffer. If null, all data units are acquired. <i>mpe_FilterSpec</i> is defined in the <i>mpe_FilterSpec</i> section. |
| | <i>queueId</i> | identifies the queue in which to send VBI events. VBI events are defined in the <i>Events</i> section. <i>mpe_EventQueue</i> is described in the <i>mpe_EventQueue</i> section. Currently, the following events are supported: |
| | MPE_VBI_EVENT_BUFFER_FULL | is sent when at least bufferSize (MPE_VBI_PARAM_DATABUFFER_SIZE) bytes are contained in the VBI session's buffer. The filtering session is stopped. |
| | MPE_VBI_EVENT_DATAUNITS_RECEIVED | is sent when at least dataUnitNotificationThreshold (MPE_VBI_PARAM_DATA_UNIT_THRESHOLD) data units are contained in the VBI session's buffer. |
| | MPE_VBI_EVENT_FILTER_AVAILABLE | is sent when memory or other resources are depleted. |
| | MPE_VBI_EVENT_FILTER_STOPPED | is sent when the filtering session has not already been put into the stopped state and <i>mpeos_vbiFilterStop()</i> is called. |
| | MPE_VBI_EVENT_FIRST_DATAUNIT | is sent when a data unit is written to a VBI session's empty buffer. |

| | |
|--------------------------------------|---|
| | MPE_VBI_EVENT_OUT_OF_MEMORY is sent when memory or other resources are depleted. The filtering session is stopped. |
| | MPE_VBI_EVENT_SOURCE_CLOSED is sent when the filtering session is stopped due to termination of the associated decode session or other resource required for VBI data acquisition. |
| | MPE_VBI_EVENT_SOURCE_SCRAMBED is sent when the VBISource becomes scrambled - preventing extraction of VBI data. MPE_VBI_EVENT_SOURCE_SCRAMBED does not stop the filtering session. |
| | MPE_VBI_EVENT_UNKNOWN indicates an unknown event occurred. |
| <i>act</i> | is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous events are sent to the queue specified by <i>queueId</i> (via <code>mpeos_eventQueueSend()</code>), the event should pass this <i>act</i> pointer in the <i>optionalEventData2</i> field. This is the Event Dispatcher (ED) handle if ED is being used. |
| <i>dataFormat</i> | specifies the data format to be retrieved from the VBI data source. <code>mpe_VBIDataFormat</code> is defined in the <i>mpe_VBIDataFormat</i> section. |
| <i>dataUnitSize</i> | specifies the data unit size, in bits, to be used for <i>dataFormat</i> MPE_VBI_FORMAT_UNKNOWN. <i>dataUnitSize</i> is ignored for all other MPE_VBI_FORMAT values. |
| <i>bufferSize</i> | specifies the size of the buffer to be used for storing retrieved VBI data, in bytes. |
| <i>dataUnitNotificationThreshold</i> | specifies the number of received data units that trigger a MPE_VBI_EVENT_DATAUNITS RECEIVED event. If 0, this notification is disabled. |
| <i>flags</i> | specifies the extended options for setting the VBI filter. Currently, <i>flags</i> is set to 0. |
| <i>filterSession</i> | is an output pointer to a location to store the filtering session handle used to identify the filter in event notifications and any subsequent operations. <code>mpe_VBIFilterSession</code> is defined in the <i>mpe_VBIFilterSession</i> section. |

| | |
|-----------------------|---|
| value returned | If the call is successful, <code>mpeos_vbiFilterStart()</code> should return MPE_SUCCESS. Otherwise, it should return one of the following error codes: |
| MPE_EINVAL | indicates at least one input parameter to the function has an invalid value. |
| MPE_ENOMEM | indicates the function failed due to insufficient memory resource. |

MPE_VBI_ERROR_FILTER_NOT_AVAILABLE

indicates insufficient filtering resources for the requested filter specification.

MPE_VBI_ERROR_SOURCE_CLOSED

indicates the specified filter source is no longer active.

MPE_VBI_ERROR_SOURCE_SCRAMBLED

indicates the specified filter source cannot be unscrambled.

description

mpeos_vbiFilterStart() should initiate VBI data acquisition and filtering. A buffer of the designated buffer size should be allocated for the duration of the VBI filtering. VBI data of the designated format, which conforms to the designated filter, should be written into the buffer as it arrives.

When the first data unit is placed in the VBI session's data buffer, a MPE_VBI_EVENT_FIRST_DATAUNIT should be sent to the queue associated with the filter session. If non-zero, MPE_VBI_EVENT_DATAUNITS RECEIVED should be sent each time dataUnitNotificationThreshold data units are received.

A variety of conditions may stop the filter session including mpeos_vbiFilterStop() (MPE_VBI_EVENT_FILTER_STOPPED), closing of the VBI source (MPE_VBI_EVENT_SOURCE_CLOSED), or resource depletion (MPE_VBI_EVENT_OUT_OF_MEMORY).

If the buffer is filled, filtering should terminate and an MPE_VBI_EVENT_BUFFER_FULL should be sent to the associated event queue.

related function(s)

mpeos_vbiFilterStop

mpeos_vbiFilterStop

stops the designated filter.

syntax `mpe_Error mpeos_vbiFilterStop(
 mpe_VBIFilterSession filterSession);`

parameter(s) `filterSession` specifies the filter. `filterSession` must have been returned by a previous call to `mpeos_vbiFilterStart()`.
`mpe_VBIFilterSession` is defined in the `mpe_VBIFilterSession` section.

value returned If the call is successful, `mpeos_vbiFilterStop()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:

`MPE_VBI_ERROR_INVALID_FILTER_SESSION`
indicates `filterSession` is invalid.

description `mpeos_vbiFilterStop()` should stop the designated filter. If the filter has not already been put into the stopped state, a `MPE_VBI_EVENT_FILTER_STOPPED` event should be put in the event queue associated with `filterSession`. Any matched data in the filter's data buffer should be retained until `mpeos_vbiFilterRelease()` is called.

related function(s) `mpeos_vbiFilterRelease`
`mpeos_vbiFilterStart`

mpeos_vbiGetParam

gets the designated parameter from the VBI filtering subsystem.

syntax `mpe_Error mpeos_vbiGetParam(mpe_VBIParameter parameter, uint32_t option1, uint32_t option2, uint32_t *outval);`

| | | |
|----------------------------|---|--|
| parameter(s) | <i>parameter</i> | specifies the offset to read from within the VBI buffer. <i>mpe_VBIParameter</i> is defined in the <i>mpe_VBIParameter</i> section. Currently, the following values are defined for <i>parameter</i> : |
| | <code>MPE_VBI_PARAM_SCTE20LINE21_CAPABILITY</code> | indicates line 21 VBI data retrieval in MPEG picture headers as defined in ANSI/SCTE 20. |
| | <code>MPE_VBI_PARAM_SCTE21LINE21_CAPABILITY</code> | indicates line 21 VBI data retrieval in MPEG picture headers as defined in ANSI/SCTE 21. |
| | <code>MPE_VBI_SEPARATED_FILTERING_CAPABILITY</code> | indicates the separated-filtering data format is supported. |
| | <code>MPE_VBI_MIXED_FILTERING_CAPABILITY</code> | indicates the mixed-filtering data format is supported. |
| <i>option1</i> | | specifies additional data. Currently, <i>option1</i> is not used. |
| <i>option2</i> | | specifies additional data. Currently, <i>option2</i> is not used. |
| <i>outval</i> | | is an output pointer for determining if the capability specified by <i>parameter</i> is supported. If the associated capability is supported, <i>outval</i> is set to 1. Otherwise, <i>outval</i> is set to 0. |
| value returned | | If the call is successful, <code>mpeos_vbiGetParam()</code> should return <code>MPE_SUCCESS</code> . Otherwise, it should return the following error code: |
| | <code>MPE_EINVAL</code> | indicates that <i>parameter</i> is unsupported for this operation. |
| description | | <code>mpeos_vbiGetParam()</code> should get the top-level subsystem parameters from the VBI filtering subsystem for the VBI session. |
| related function(s) | | <code>mpeos_vbiFilterSetParam</code> |

mpeos_vbilinit

initializes the VBI filtering system.

syntax mpe_Error mpeos_vbiInit(void);

parameter(s) none

value returned If the call is successful, mpeos_vbiInit() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_VBI_ERROR indicates initialization failed.

description mpeos_vbiInit() should initialize the VBI filtering system. mpeos_vbiInit() should only be called once and no other VBI function should be called until mpeos_vbiInit() returns with an MPE_SUCCESS return value.

related function(s) mpeos_vbiShutdown

mpeos_vbiRegisterAvailability

registers for available VBI filters notification.

syntax mpe_Error mpeos_vbiRegisterAvailability(
 mpe_EventQueue queueId ,
 void * act);

| | | |
|---------------------|----------------|---|
| parameter(s) | <i>queueId</i> | identifies the queue in which to send the VBI events. VBI events are defined in the <i>Events</i> section. <i>mpe_EventQueue</i> is described in the <i>mpe_EventQueue</i> section. Currently, the follow events are supported: |
| | | MPE_VBI_EVENT_BUFFER_FULL is sent when at least bufferSize (MPE_VBI_PARAM_DATABUFFER_SIZE) bytes are contained in the VBI session's buffer. The filtering session is stopped. |
| | | MPE_VBI_EVENT_DATAUNITS_RECEIVED is sent when at least dataUnitNotificationThreshold (MPE_VBI_PARAM_DATA_UNIT_THRESHOLD) data units are contained in the VBI session's buffer. |
| | | MPE_VBI_EVENT_FILTER_AVAILABLE is sent when memory or other resources are depleted. |
| | | MPE_VBI_EVENT_FILTER_STOPPED is sent when the filtering session has not already been put into the stopped state and <i>mpeos_vbiFilterStop()</i> is called. |
| | | MPE_VBI_EVENT_FIRST_DATAUNIT is sent when a data unit is written to a VBI session's empty buffer. |
| | | MPE_VBI_EVENT_OUT_OF_MEMORY is sent when memory or other resources are depleted. The filtering session is stopped. |
| | | MPE_VBI_EVENT_SOURCE_CLOSED is sent when the filtering session is stopped due to termination of the associated decode session or other resource required for VBI data acquisition. |
| | | MPE_VBI_EVENT_SOURCE_SCRAMBLED is sent when the VBISource becomes scrambled - preventing extraction of VBI data. MPE_VBI_EVENT_SOURCE_SCRAMBLED does not stop the filtering session. |
| | | MPE_VBI_EVENT_UNKNOWN indicates an unknown event occurred. |

act is an input pointer to an Asynchronous Completion Token (ACT). When the asynchronous events are sent to the queue specified in *queueId* (via `mpeos_eventQueueSend()`), the event should pass this *act* pointer in the *optionalEventData2* field.

value returned If the call is successful, `mpeos_vbiRegisterAvailability()` should return `MPE_SUCCESS`. Otherwise, it should return one of the following error codes:

`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

`MPE_ENOMEM` indicates insufficient memory.

description `mpeos_vbiRegisterAvailability()` should register an event queue to be notified when filters or other dependent resources are available and under certain other platform-specific conditions. Following successful registration, an event should be sent to the given event queue signaling filter availability when applicable. When an event is sent (and received) it should be composed of the following information:

- ◆ *eventId* identifies the specific VBI event, `MPE_VBI_FILTERS_AVAILABLE`.
- ◆ *optionalEventData1* is not being used by VBI.
- ◆ *optionalEventData2* identifies the Asynchronous Completion Token (ACT) to be passed in after the event is delivered to the queue.
- ◆ *optionalEventData3* is not being used by VBI.

related function(s) `mpeos_vbiUnregisterAvailability`

mpeos_vbiShutdown

shuts down the VBI filtering system.

syntax mpe_Error mpeos_vbiShutdown(void);

parameter(s) none

value returned If the call is successful, mpeos_vbiShutdown() should return MPE_SUCCESS. Otherwise, it should return the following error code:

MPE_VBI_ERROR indicates initialization failed.

description mpeos_vbiShutdown() should shutdown the VBI filtering system. mpeos_vbiShutdown() should only be called once. No other VBI function should be called after mpeos_vbiShutdown() returns, regardless of the return code. The VBI buffer filtering subsystem should release as many resources as possible. But outstanding filters do not need to be issued MPE_VBI_EVENT_SOURCE_CLOSED or any other events on shutdown.

related function(s) mpeos_vbiInit

mpeos_vbiUnregisterAvailability

unregisters for available VBI filters notification.

syntax mpe_Error mpeos_vbiUnregisterAvailability(
 mpe_EventQueue *queueId* ,
 void * *act*);

parameter(s) *queueId* identifies the queue to unregister.
act is an input pointer to an ACT. When the asynchronous events are sent to the queue specified in *queueId* (via `mpeos_eventQueueSend()`), the event should pass this *act* pointer in the *optionalEventData2* field.

value returned If the call is successful, `mpeos_vbiUnregisterAvailability()` should return `MPE_SUCCESS`. Otherwise, it should return the following error code:
`MPE_EINVAL` indicates at least one input parameter to the function has an invalid value.

description `mpeos_vbiUnregisterAvailability()` should unregister the specified event queue and completion token pair that was previously provided on a call to `mpeos_vbiRegisterAvailability()`. After successful completion of this call, the specified event queue should no longer receive notifications of filter availability.

related function(s) `mpeos_vbiRegisterAvailability`

A Porting the Java Virtual Machine

Overview

The OCAP stack is implemented upon the standard Java APIs, however, in order to implement OCAP features correctly there are several non-standard integration points. Supporting these integration points within a JVM implementation or port is necessary to correctly implement the OCAP stack. However, lack of support should only result in diminished functionality in certain areas.

NOTE: This section is intended for someone using a VM that has not been integrated with the OCAP stack.

The integration points can be divided into the following categories:

- ◆ AWT
- ◆ File input/output
- ◆ Resource reclamation
- ◆ Application context
- ◆ Security
- ◆ Miscellaneous

Before reading this chapter, you should:

- ◆ have MPEOS ported to the platform
- ◆ have a port of the existing VM to the platform, including AWT
- ◆ be able to run OCAP within the VM without features that depend upon special integration points

After reading this chapter, you should be able to run OCAP within the VM including features that depend upon special integration points.

Integration points are defined by classes or interfaces defined within the OCAP stack. It is expected that the Java library implementation, provided by the JVM, implement these classes and interfaces. While the OCAP implementation does define these classes, these versions are not expected to be used at runtime – their presence is mainly for documentation and build-time reference purposes. It is expected that the OCAP stack is referenced on the JVM's classpath, rather than boot classpath, to ensure the JVM library's definitions of these APIs are used at runtime.

Java's Abstract Windowing Toolkit (AWT)

The OCAP implementation expects an AWT implementation to support several points of integration in order to provide support for the following features:

- ◆ OCAP event model
- ◆ OCAP focus model
- ◆ DVB graphics
- ◆ Font factory
- ◆ Graphics environment

OCAP event model

In order to enable the OCAP event model (as described in OCAP 1.0 Annex K) an AWT implementation must support the following interfaces defined within the OCAP stack:

```
com.vidiom.impl.awt.EventDispatchable
    allows an EventDispatcher to be installed which handles
    dispatching of events to applications.
    com.vidiom.impl.awt.EventDispatchable is implemented
    by the AWT Toolkit implementation.

com.vidiom.impl.awt.EventDispatcher
    modifies the AWT event dispatch mechanisms for OCAP
    applications. com.vidiom.impl.awt.EventDispatcher is an
    interface of a strategy class.
```

The EventDispatchable interface should be implemented by the AWT Toolkit implementation. This allows the OCAP implementation to provide an EventDispatcher object through which key events are routed. Rather than posting KeyEvent to the toolkit's EventQueue, an InvocationEvent that invokes the dispatcher's dispatch method with the KeyEvent should be posted. For example, a postKeyEvent(KeyEvent) method could be implemented subject to whether an EventDispatcher(dispatcher) is set or not similar to the following:

```
// Post AWT event to event queue
if (tk.dispatcher == null)
    queue.postEvent(event);
// Post an InvocationEvent() that will handle calling of
EventDispatcher
else
{
    Runnable run = new Runnable()
    {
        public void run()
        {
            tk.dispatcher.dispatch(event);
        }
    };
    queue.postEvent(new InvocationEvent(source, run));
}
```

OCAP focus model

In order to enable the OCAP focus model (as described in OCAP 1.0 25.2.2.1.1 and OCAP 1.0 Annex K) an AWT implementation must support the FocusHandler class as defined in the OCAP stack.

```
com.vidiom.impl.awt.FocusHandler
    allows for an alternate mechanism to be used to manage
    component focus.
```

The FocusHandler class provides an integration point for redirecting component focus change requests. The implementation of `java.awt.Component.requestFocus()` should be modified so it has the following code added to the front:

```
// Added by Vidiom to allow OCAP Focus Management rules
if ( com.vidiom.impl.awt.FocusHandler.requestFocus( this ) )
    return;
```

DVB graphics

The `org.dvb.ui.DVBGraphics` implementation provided by the OCAP stack is built upon the `Graphics2D` provided by the underlying AWT implementation. The OCAP stack simply adapts each instance of a `Graphics2D` object to the DVB graphics API by decorating it with a wrapper class. In order to support this, an AWT implementation must support the following interfaces defined within the OCAP stack:

```
com.vidiom.impl.awt.GraphicsAdaptable
    allows a GraphicsFactory to be installed which wraps the
    AWT implementation-provided Graphics object.
```

```
com.vidiom.impl.awt.GraphicsFactory
    modifies the AWT Graphics implementation. This can be
    used by the AWT Toolkit implementation to allow the
    native Graphics implementation to be wrapped by an
    adapter or decorator object.
```

The `GraphicsAdaptable` interface should be implemented by the AWT Toolkit implementation. This allows the OCAP implementation to provide a `GraphicsFactory` object capable of wrapping any `Graphics` object to be returned to an application.

All AWT methods which return a `Graphics` object (except those defined by `Graphics` or `Graphics2D`) should invoke the `GraphicsFactory.wrap(Graphics)` method to allow the OCAP implementation to decorate the returned instance. In general, this is limited to implementations of `Image.getGraphics()`. There is no need to wrap graphics when invoking `Component.paint(Graphics)` as this is already taken care of through the OCAP implementation.

Font factory

In order to support application-downloadable fonts, the AWT implementation must provide an implementation of `FontFactoryPeer` as defined by the OCAP stack:

```
com.vidiom.impl.awt.FontFactoryPeer  
    provides platform-specific implementation of FontFactory  
    native peer.
```

The OCAP stack will create an instance of this class for every instance of `FontFactory` created by an application.

Graphics environment

For proper integration of the HAVi implementation with the underlying AWT implementation, the AWT implementation should implement the graphics environment in the manner expected by the OCAP stack and also implement the following APIs:

```
com.vidiom.impl.awt.NativePeer  
    exposes the native handle for a native peer.
```

```
com.vidiom.impl.awt.ResizableFrame  
    provides a method for resizing to match the current bounds  
    of the #getGraphicsConfiguration  
    GraphicsConfiguration.
```

The `GraphicsEnvironment` implementation is expected to expose all of the available screens. The `GraphicsDevice` implementation is expected to only expose the current configuration, as there is no means to modify the current configuration. In other words, `getDefaultConfiguration()` and `getConfiguration()` return a single configuration that corresponds to the current configuration at the time of the call. It is expected that the `GraphicsDevice` implementation implements the `NativePeer` interface, allowing HAVi to correlate MPE graphics devices with AWT graphics devices.

The `ResizableFrame` is used to implement the top-level window for a given graphics device. The OCAP stack expects the `updateBounds` method, when invoked, to update the bounds of the window to match the current graphics device configuration. This is used to ensure the top-level frame is full-screen following HAVi configuration changes.

File input/output (IO)

In OCAP, file authentication is integrated into the Java File IO model. Also, OCAP requires stack management of persistent storage, including monitoring of quotas and file purging when necessary. In order to support this, the following standard Java classes must, at a minimum, be modified to include integration with APIs defined by the OCAP stack:

- ◆ `java.io.FileDescriptor`
- ◆ `java.io.FileInputStream`
- ◆ `java.io.File`
- ◆ `java.io.FileOutputStream`
- ◆ `java.io.RandomAccessFile`
- ◆ `java.util.zip.ZipFile`

The OCAP stack defines the following interfaces and classes which should be implemented by the JVM implementation:

- `com.vidiom.impl.io.AsyncLoadCallback`
performs an asynchronous load of the file referenced by path. This method is used to support `DSMCCObject.asynchronousLoad()`.
- `com.vidiom.impl.io.AsyncLoadHandle`
cancels an in-progress asynchronous load. A unique instance is returned from a call to `FileSys.asynchronousLoad()`.
- `com.vidiom.impl.io.DefaultFileSys`
specifies the default implementation of the `FileSys` interface. Most methods call directly to the native file system on the host.
- `com.vidiom.impl.io.DefaultOpenFile`
specifies the `com.vidiom.impl.io.OpenFile` implementation for accessing files directly through the native MPE APIs.
- `com.vidiom.impl.io.DefaultWritableFileSys`
supports direct access to the native "write" output support.
- `com.vidiom.impl.io.FileSys`
represents a file system used to retrieve file data.
- `com.vidiom.impl.io.FileSysAccessor`
obtains the `FileSys` instance from a `File` object.
- `com.vidiom.impl.io.FileSysHelper`
maintains the `FileSysAccessor` instance
- `com.vidiom.impl.io.FileSysImpl`
specifies the default abstract implementation of the `FileSys` interface. All methods either do nothing or return default values, namely `null` or `false`.

`com.vidiom.impl.io.FileSysManager`
allows the implementation to retrieve the appropriate
`FileSys` instance for reading and the appropriate
`WriteableFileSys` instance for writing to a path. Also allows
the implementation to update the `FileSysMgr` for the entire
system.

`com.vidiom.impl.io.FileSysMgr`
specifies a class that implements this interface can be
installed as the default file manager for the implementation.
The class returns the correct `FileSys` or `WriteableFileSys`
instances based on the supplied path.

`com.vidiom.impl.io.OpenFile`
represents an open file and is the base class for all open file
classes used to obtain file data. An instance of this interface
is returned from a call to `FileSys.open()` and
`FileSys.openClass()`.

`com.vidiom.impl.io.StorageMediaFullException`
represents a storage media full error for a storage device.

`com.vidiom.impl.io.WritableFileSys`
gets a `WriteableFileSys` object based on a path name.

The standard classes from the `java.io` packages listed above should be modified to use the classes from the `com.vidiom.impl.io` package for IO instead of using native methods to directly get file data. The normal sequence of events to read file data is to first obtain an instance of a `FileSys` from the `FileSysManager` based on the file path. This is normally done on the construction of a `java.io.File` object. Then when file data is needed, an `OpenFile` instance can be obtained from the `open()` method on the `FileSys`. The methods on the `OpenFile` instance can be used to obtain the file data.

A similar process is used when file output is performed. An instance of a `WritableFileSys` is obtained from the `FileSysManager` and that instance is used to perform the file output operations.

Resource reclamation

OCAP requires that prior to throwing `OutOfMemoryError` some general steps be taken including generation of `ResourceDepletionEvents` and the destruction of lower-priority applications. In order to fully support this a JVM integration point has been defined in the following form of a `Thread` subclass:

`com.vidiom.impl.java.ReclaimThread`

is an abstract base class that should be extended to implement resource reclamation procedures applied before throwing `OutOfMemoryError`. A subclass should override `reclaimMemory(int, long)` to implement the resource reclamation process.

The `ReclaimThread` subclass of `Thread` is sub-classed by the OCAP stack, which overrides the `reclaimMemory()` method and is responsible for starting the thread. The execution of the thread, as provided by the `run()` method, is expected to be defined by the JVM.

Prior to throwing an `OutOfMemoryError` in response to a failed allocation request, the JVM implementation should invoke the `reclaimMemory()` method defined by the stack-created instance of `ReclaimThread`. `ReclaimThread` should be called repeatedly for each potentially-failing memory allocation request until zero is returned. The following arguments should be passed to the `reclaimMemory()` method:

- ◆ The level of escalation. This should be 1 (one) on the first invocation and the return value of previous invocations on subsequent invocations until 0 (zero) is returned.
- ◆ The thread-specific context identifier. This value may be acquired from MPE-specific thread local storage.

After invoking the `reclaimMemory()` method the implementation is expected to follow with appropriate garbage collection and finalization cycle(s) and then to retry allocation. Once 0 (zero) is returned and allocation cannot be satisfied, then `OutOfMemory` should be thrown.

Application context

The OCAP stack maintains its own concept of separate applications (Xlets), separate from anything in the JVM implementation. For a large part, this is built upon existing Java concepts (for example, ThreadGroup), but extension or modification of the underlying Java libraries is necessary in some cases to reflect this concept. The OCAP stack expects the JVM to support the following API interfaces:

`com.vidiom.impl.java.AppContext`

represents the set of information specific to a given application. This class is similar to the `AppContext` used within a J2SE implementation to provide for applet-specific information. Instances of `AppContext` can be retrieved using the `AppContext.Factory.getInstance()` factory method. By default, only one `AppContext` is ever returned. However, this can be overridden by implementing an instance of `AppContext.Factory` and specifying that it be used via `AppContext.Factory.setFactory(com.vidiom.impl.java.AppContext.Factory)`.

`com.vidiom.impl.java.AppContext.Factory`

specifies an interface for an `AppContext` factory.

It is expected that the Java implementation be updated to use the `AppContext` API. At this time, the `AppContext` API is used only to modify the set of Java System properties visible to specific applications.

-
- ◆ **NOTE:** This mechanism is not used to restrict property access that is managed via Java fine-grained security controls.
-

The `java.lang.System.getProperty(String)` method should call out to the current `AppContext` as follows:

```
return AppContext.Factory.getInstance()  
    .getProperty(key, systemProperties.getProperty(key));
```

The `java.lang.System.getProperties()` method should call out to the current `AppContext` as follows:

```
return AppContext.Factory.getInstance()  
    .getProperties(systemProperties);
```

It is assumed that `java.lang.System.getProperty(String, String)` is implemented in terms of `java.lang.getProperty(String)`.

Security

All classes that make up the OCAP stack should be granted AllPermissions by the system class loader. This can be accomplished via the security policy configuration file, `java.policy`, with the following entry:

```
grant {  
    permission java.security.AllPermission;  
};
```

Miscellaneous

The following topics are included in this section:

| | |
|---|-----------|
| Class file verification PBP compliance | Time zone |
|---|-----------|

Class file verification

It is recommended that the OCAP stack classes be considered already-vetted when it comes to class file verification so as to improve stack start-up time.

Time zone

It is recommended that the `java.util.TimeZone` implementation either not cache the known default `TimeZone` object or hold off caching the object until after a time zone other than GMT is seen. This is suggested in order to allow for delayed time zone acquisition (for example, from the network).

PBP compliance

It is expected that the following Java system properties be defined, declaring levels of PBP compliance and/or restrictions:

`java.awt.AlphaComposite.SRC_OVER`
has a value of `false` and is full-composition supported.

`java.awt.event.KeyEvent.isRestricted`
has a value of `true`.

`java.awt.event.KeyEvent.supportMask`
has a value of 7 and uses UP/DOWN/LEFT/RIGHT, 0-9.

`java.awt.Component.setCursor.isRestricted`
has a value of `true`.

`java.awt.Frame.setSize.isRestricted`
has a value of `true`.

`java.awt.Frame.setResizable.isRestricted`
has a value of `true`.

`java.awt.Frame.setLocation.isRestricted`
has a value of `true`.

`java.awt.Frame.setState.isRestricted`
has a value of `true`.

`java.awt.Frame.setTitle.isRestricted`
has a value of `true`.

`java.awt.event.MouseEvent.isRestricted`
has a value of `true`.

`java.awt.event.MouseEvent.supportLevel`
has a value of 0.

Glossary

| | |
|--|--|
| abstract windowing toolkit (AWT) | The AWT is Java's platform-independent windowing, graphics, and user-interface widget toolkit. The AWT is part of the Java Foundation Classes, the standard API for providing a graphical user interface (GUI) for a Java program. |
| AC | See <i>alternating current (AC)</i> . |
| ACT | See <i>asynchronous completion token (ACT)</i> . |
| Advanced Television Systems Committee | The ATSC is the group that helped develop the new digital television standard for the United States. |
| AIT | See <i>application information table (AIT)</i> . |
| alpha red green blue (ARGB) | ARGB converts code to color values |
| alternating current (AC) | AC is an electrical current that reverses direction in varying cycles. |
| APDU | See <i>application protocol data unit (APDU)</i> . |
| application information table (AIT) | An AIT is used to signal in-band applications that are bound to a broadcast service. |
| API | See <i>application programming interface (API)</i> . |
| application programming interface (API) | An API is a series of functions that applications can use to make the operating system perform specific tasks. APIs let you program without having intimate knowledge of the device or software to which you are sending commands. An API can consist of classes, function calls, subroutine calls, descriptive tags, etc. |
| application protocol data unit (APDU) | An APDU is the communication unit between a reader and a POD. The structure of an APDU is defined by the ISO 7816 standards. There are two categories of APDUs: command APDUs (mandatory 5-byte header) and response APDUs (mandatory 2-byte status word). |

| | |
|---|---|
| ARGB | See <i>alpha red green blue (ARGB)</i> . |
| asynchronous completion token (ACT) | ACT is an additional piece of information needed to pass the event from native (MPEOS/MPE/JIN land) into java (OCAP stack). |
| ATSC | See <i>Advanced Television Systems Committee</i> . |
| AWT | See <i>abstract windowing toolkit (AWT)</i> . |
| BFS | See <i>broadcast file system (BFS)</i> . |
| binary phase-shift keying (BPSK) | BPSK is a binary digital modulation scheme that conveys data by changing the phase of a reference signal. |
| broadcast file system (BFS) | The BFS is a data carousel system by which application data can be stored on an application server and transmitted frequently to the set-top boxes for application use. It controls a file and directory structure broadcast repeatedly in a carousel over the cable network. |
| BPSK | See <i>binary phase-shift keying (BPSK)</i> . |
| BSD | See <i>Berkeley software distribution (BSD)</i> . |
| Berkeley software distribution (BSD) | The BSD is the name of the UNIX derivative distributed in the 1970s from the University of California, Berkeley. The name is also used collectively for the modern descendants of these distributions. |
| CD | See <i>compact disc (CD)</i> . |
| central processing unit (CPU) | ACPU interprets and carries out, or processes, instructions and data contained in the software. |
| compact disc (CD) | A CD is used to store digital data. |
| CPU | See <i>central processing unit (CPU)</i> . |
| CRC | See <i>cyclical redundancy check (CRC)</i> . |
| cyclical redundancy check (CRC) | A CRC is a type of check value designed to catch most transmission errors. |
| data-over-cable service interface specification (DOCSIS) | DOCSIS is an international standard for communications and operation support interface requirements for a data over cable system. |
| DAVIC | See <i>digital audio video council (DAVIC)</i> . |
| DCII | See <i>Digicipher II</i> . |
| DDB | See <i>downloadable data blocks (DDB)</i> . |
| decoder format conversion (DFC) | Converts the initial transferred frame into the specified output frame format. |
| development system | A development system refers to the computer being used to port the OCAP stack. |

| | |
|---|---|
| DFC | See <i>decoder format conversion (DFC)</i> . |
| DHCP | See <i>dynamic-host configuration protocol (DHCP)</i> . |
| Digicipher II | DCII is Motorola's MPEG-2 based distribution system. It is used by about 70% of cable channels in North America to distribute their video to cable headends, other satellite companies like DirectTV and Dish Network and also available to backyard dish owners via Motorola's 4DTV satellite receiver products. |
| digital audio video council (DAVIC) | The DAVIC establishes the industry standard for end-to-end interoperability of broadcast and interactive digital audio-visual information. |
| digital storage media - command and control (DSM-CC) | DSM-CC is a toolkit for developing control channels associated with MPEG-1 and MPEG-2 streams. |
| digital video recorder (DVR) | A DVR records television shows to a hard disk in digital format. |
| digital visual interface (DVI) | A DVI is a video connector designed to maximize the quality of digital display devices. |
| DII | See <i>dynamic invocation interface (DII)</i> . |
| DLL | See <i>dynamic link library (DLL)</i> . |
| DNS | See <i>domain name server (DNS)</i> . |
| DOCSIS | See <i>data-over-cable service interface specification (DOCSIS)</i> . |
| DOCSIS service gateway (DSG) | DSG is a standard describing how out-of-band data is delivered to a set-top box. |
| domain name server (DNS) | A DNS relates an Internet domain name—such as www.vidiom.com —to an Internet Protocol (IP) address. See also <i>Internet protocol (IP)</i> . |
| downloadable data blocks (DDB) | Object carousels are made up of sets of DDBs, which contain up to about 4K of data each. |
| dynamic invocation interface (DII) | A DII is used to send requests from the client application through synchronous and one-way communication. |
| dynamic link library (DLL) | A DLL is a collection of sub-programs used to develop software. |
| DSG | See . |
| DSM-CC | See <i>digital storage media - command and control (DSM-CC)</i> . |
| DVI | See <i>digital visual interface (DVI)</i> . |
| DVR | See <i>digital video recorder (DVR)</i> . |
| dynamic-host configuration protocol (DHCP) | DHCP is the internet standard for assigning <i>Internet protocol (IP)</i> addresses dynamically to <i>Internet protocol (IP)</i> host devices. |

| | |
|--|--|
| EAS | See <i>emergency alert system (EAS)</i> . |
| EIA | See <i>Electronics Industry Association (EIA)</i> . |
| Electronics Industry Association (EIA) | The EIA provides the standards for the electronics industry. |
| electronic programming guide (EPG) | An EPG is an on-screen guide to scheduled broadcast television programs. |
| emergency alert system (EAS) | EAS is used to send messages over radio and television stations. |
| EPG | See <i>electronic programming guide (EPG)</i> . |
| Esmertec Virtual Machine (EVM) | The EVM is the virtual machine used in the OCAP stack. |
| EVM | See <i>Esmertec Virtual Machine (EVM)</i> . |
| exclusive or (XOR) | XOR determines the result depending on the number of TRUE operands. If there are an odd number of TRUE operands, then the result is TRUE, Otherwise, it is FALSE. |
| FDC | See <i>forward data channel (FDC)</i> . |
| forward data channel (FDC) | The FDC is in the RF range (70-130 MHz) and is used to deliver data from the head-end to the set-top box. |
| GDI | See <i>graphics device interface (GDI)</i> . |
| GMT | See <i>Greenwich Mean Time (GMT)</i> . |
| graphics device interface (GDI) | A GDI draws lines and curves, renders fonts, and handles palettes. |
| Greenwich Mean Time (GMT) | GMT is used as an international time reference. |
| extended application information table (XAIT) | An XAIT is used to signal out-of-band applications that are not bound to a broadcast service. |
| HAVi | See <i>home audio video interoperability (HAVi)</i> . |
| home audio video interoperability (HAVi) | HAVi is a software-based on the Institute of Electrical and Electronic Engineers (IEEE) 1394 standard that can make it possible for connections and integration of home entertainment devices, such as set-top boxes with other electronic appliances on a network, particularly a home networking system. |
| host device | A host device is an OpenCable-compliant cable receiver (for example, set-top box or cable-ready TV). |
| IEC | See <i>International Electrotechnical Commission (IEC)</i> . |
| International Electrotechnical Commission (IEC) | The IEC is an international standards organization dealing with electrical, electronic, and related technologies. |

| | |
|---|---|
| International Organization for Standardization (ISO) | The ISO is an international standards body. |
| IP | See <i>Internet protocol (IP)</i> . |
| Internet protocol (IP) | IP refers to the network layer (layer 3) of the Open Systems Interconnect (OSI) Internet protocol, providing connectionless datagram service. It describes the computer network protocol—analogous to written and verbal languages—that all machines on the Internet must know so that they can communicate with one another. It also is the network layer of Transmission Control Protocol/Internet Protocol (TCP/IP) which controls the flow of data packets. |
| IPPV | See <i>impulse-pay-per-view (IPPV)</i> . |
| impulse-pay-per-view (IPPV) | IPPV is a movie rental service similar to pay-per-view (PPV). However, IPPV allows the customer to purchase the right to view the movie or event through an on-screen interface. |
| ISO | See <i>International Organization for Standardization (ISO)</i> . |
| JAR | See <i>Java Archive (JAR)</i> . |
| Java Archive (JAR) | A JAR file format enables you to bundle multiple files into a single archive file. Typically, a JAR file will contain the class files and auxiliary resources associated with applets and applications. |
| Java media framework (JMF) | The JMF enables audio, video, and other time-based media to be added to applications and applets built on Java technology. |
| Java native interface (JNI) | The JNI is the Java native programming interface and is included in the JDK. Programs written using the JNI are completely portable across all platforms. |
| Java virtual machine (JVM) | The JVM interprets compiled Java binary code (called byte-code) for a computer's processor (or "hardware platform") so that it can perform a Java program's instructions. |
| JMF | See <i>Java media framework (JMF)</i> |
| JNI | See <i>Java native interface (JNI)</i> . |
| JVM | See <i>Java virtual machine (JVM)</i> . |
| man machine interface (MMI) | A MMI is another term for user interface. For the purpose of this document, the term MMI specifies the protocol used over the POD/host interface to enable the POD device to display messages on the television. |
| media storage volume (MSV) | A MSV is storage space requested by an application which is typically used for media storage, but can also be used for general purposes (data file). |
| MHEG | See <i>Multimedia and Hypermedia information coding Expert Group (MHEG)</i> . |
| MIME | See <i>multipurpose internet mail extensions (MIME)</i> . |

| | |
|---|---|
| MMI | See <i>man machine interface (MMI)</i> . |
| MPE | See <i>multimedia platform environment (MPE)</i> . |
| Multimedia and Hypermedia information coding Expert Group (MHEG) | The MHEG is a standard devised for the middleware of digital teletext services in the United Kingdom. |
| multimedia platform environment (MPE) | The MPE middleware layer provides much of the native multimedia functionality of the host device client. MPE is a collection of platform independent APIs and associated native managers that provide multimedia-related services similar to the Java managers. The MPE middleware layer provides the following for multimedia resources and events: access, registration, logging, coordination, and delivery. |
| multipurpose internet mail extensions (MIME) | MIME is an Internet Standard email format. |
| MPEG | See <i>moving picture experts group (MPEG)</i> . |
| MPEG-1 | See <i>moving picture experts group - video CD (MPEG-1)</i> . |
| MPEG-2 | See <i>moving picture experts group - broadcast signals (MPEG-2)</i> . |
| moving picture experts group (MPEG) | The MPEG develops standards for digital video and digital audio compression. |
| moving picture experts group - video CD (MPEG-1) | MPEG-1 is the audio and video coding standards agreed on by MPEG (Moving Picture Experts Group). MPEG-1 video is used by the video CD format. |
| moving picture experts group - broadcast signals (MPEG-2) | MPEG-2 is typically used to encode audio and video for broadcast signals, including direct broadcast satellite and cable TV. |
| MPEOS | See <i>multimedia platform environment operating system (MPEOS)</i> . |
| multimedia platform environment operating system (MPEOS) | MPEOS is an operating system/middleware porting layer that provides extensive support for the various operating system and multimedia-oriented services of the client environment. To achieve a high degree of portability, MPEOS also provides within its own domain an additional thin abstraction layer porting library. |
| multiple systems operator (MSO) | MSO is a cable operator. |
| MSO | See <i>multiple systems operator (MSO)</i> . |
| MSV | See <i>media storage volume (MSV)</i> . |
| national television system(s) committee (NTSC) | NTSC is the system, that includes VBI lines, which delivers a data bit sequence in analog video. |
| native layer | The native layer refers to the MPEOS and MPE layers and can imply binary or source depending on how it is used. |

| | |
|---|--|
| network information table (NIT) | The NIT includes the Carrier Definition Subtable and the Modulation Mode Subtable. For more information, refer to the SCTE 65 Standard. |
| network service access point (NSAP) | The NSAP is used by the object carousel as an access point to the network service. |
| network text table (NTT) | The NTT includes the Source Name Subtable. For more information, refer to the SCTE 65 Standard. |
| NIT | See <i>network information table (NIT)</i> . |
| NPT | See <i>normal play time (NPT)</i> . |
| non-volatile random access memory (NVRAM) | NVRAM is a type of memory which does not lose its information when powered off. |
| normal play time (NPT) | NPT is a time code that is embedded in a special descriptor in an MPEG-2 private section. |
| NSAP | See <i>network service access point (NSAP)</i> . |
| NTT | See <i>network text table (NTT)</i> . |
| NTSC | See <i>national television system(s) committee (NTSC)</i> . |
| NVRAM | See <i>non-volatile random access memory (NVRAM)</i> . |
| OCAP | See <i>OpenCable application platform (OCAP)</i> . |
| OCAP stack | The OCAP stack is the MPEOS, MPE, Java layers, and test applications after they have been ported and compiled. |
| OEM | See <i>original equipment manufacturer (OEM)</i> . |
| offset quadrature phase-shift keying (OQPSK) | OQPSK is an offset-quadrature digital modulation scheme that conveys data by changing the phase of a reference signal. |
| on-screen display (OSD) | An OSD is an image superimposed on a screen picture, commonly used by televisions, VCRs, and DVD players to display information such as volume, channel, and time. |
| OOB | See <i>out-of-band (OOB)</i> . |
| OpenCable application platform (OCAP) | OCAP is a software interface specification that completely defines the OpenCable host software interface that executes OpenCable portable, interactive applications, and services. OCAP is a separate effort from the Advanced Television Enhancement Forum (ATVEF) content specification, but OCAP does call for support and extension of ATVEF as a part of the presentation engine (PE) requirements. |
| OQPSK | See <i>offset quadrature phase-shift keying (OQPSK)</i> . |
| original equipment manufacturer (OEM) | An OEM is when one company manufactures a brand for another company. An example would be when one brand and/or model of VCR is manufactured at the VCR plant(s) of another brand of VCR. |

- OSD** See *on-screen display (OSD)*.
- out-of-band (OOB)** OOB is a dedicated channel that is always tuned and can be used for two-way traffic.
- PAT** See *program association table (PAT)*
- packet identification (PID)** A PID is a unique integer value used to identify elementary streams of a program in a single- or multi-program MPEG-2 stream.
- Pay-Per-View (PPV)** PPV is the system in which television viewers can purchase events to be seen on TV.
- PCR** See *program clock reference (PCR)*.
- permission request file (PRF)** A PRF is used to signal application-request privileges.
- PFR** See *portable font resource (PFR)*.
- PID** See *packet identification (PID)*.
- PMT** See *program map table (PMT)*.
- POD** See *point-of-deployment device (POD)*.
- point-of-deployment device (POD)** A POD allows a retail cable set-top box to be portable across a variety of different cable system headends by standardizing the communication between individual addressable POD modules and the connected set-top boxes. It also allows the new generation of digital cable-ready devices to connect to cable television without a set-top box.
- POSIX** See *portable operating system interface on Unix (POSIX)*.
- portable font resource (PFR)** PFR contains a set of glyph shapes associated with a character code. The PFR format is platform-independent and is intended to facilitate accurate rendering of fonts in all environments whether or not they have the required fonts already installed.
- portable operating system interface on Unix (POSIX)** POSIX is the application program interface for software designed to run on variants of the Unix operating system.
- porting kit** The porting kit includes all files/directories included in the installation.
- PPV** See *Pay-Per-View (PPV)*.
- PRF** See *permission request file (PRF)*.
- program association table (PAT)** A PAT provides the correspondence between a program number (a numeric label associated with a program) and the PID value of the transport stream packets which carry the program definition.
- program clock reference (PCR)** The PCR is used to help decoders handle synchronization issues between audio/video, closed caption, etc.

| | |
|--|---|
| program map table (PMT) | A PMT provides the most important information contained in the MPEG Program Map Table (PMT). A six-digit string of characters and numbers (example: 0x00B3) defines an MPEG program and it is made up of video packets, audio packets, and clock (synchronization) information. |
| program specific information (PSI) | PSI data contains four tables: Program Association Table (PAT), Conditional Access Table (CAT), Program Map Table (PMT) and Network Information Table (NIT). |
| PSI | See <i>program specific information (PSI)</i> . |
| QAM | See <i>quadrature amplitude modulation (QAM)</i> . |
| quadrature amplitude modulation (QAM) | QAM is a modulation scheme that conveys data by modulating the amplitude of two carrier waves. |
| QPSK | See <i>quadrature phase-shift keying (QPSK)</i> . |
| quadrature phase-shift keying (QPSK) | QPSK is a digital modulation scheme that conveys data by changing the phase of a reference signal. |
| RDC | See <i>reverse data channel (RDC)</i> . |
| RDD | See <i>revision detection descriptor (RDD)</i> . |
| red, green, blue color model (RGB) | RGB combines red, green, and blue in various ways to create other colors. |
| reverse data channel (RDC) | The RDC is in the RF range (5-40 MHz) and is used to deliver data from the set-top box to the head-end. |
| revision detection descriptor (RDD) | RDD information is embedded in the OOB service information tables and notifies the multiplier of a change to the data. |
| RGB | See <i>red, green, blue color model (RGB)</i> . |
| ROM | See <i>read-only memory (ROM)</i> . |
| read-only memory (ROM) | ROM is memory that is read-only. For example, it typically stores instructions used to start up the computer. |
| SAS | See <i>specific application support (SAS)</i> . |
| SDK | See <i>software development kit (SDK)</i> . |
| secure network file system (SNFS) | SNFS is a data carousel system by which application data can be stored on an application server and transmitted frequently to the set-top boxes for application use. It controls a file and directory structure broadcast repeatedly in a carousel over the cable network. |
| short-form virtual channel table (SVCT) | The SVCT includes the Virtual Channel Map, Defined Channels Map, and Inverse Channel Map. For more information, refer to the SCTE 65 Standard. |

| | |
|--|--|
| software development kit (SDK) | An SDK is a platform specific programming package that enables a programmer to develop applications. Typically, an SDK includes one or more APIs, programming tools, and documentation. |
| SNFS | See <i>secure network file system (SNFS)</i> . |
| specific application support (SAS) | SAS is used for private communications between vendor-specific POD module applications and private host applications. |
| SVCT | See <i>short-form virtual channel table (SVCT)</i> . |
| TCP | See <i>transmission control protocol (TCP)</i> . |
| transmission control protocol (TCP) | TCP/IP is the most common protocol used for communication between computers on a network. It was originally developed by the US Department of Defense for a worldwide communications network that eventually developed into the Internet. |
| UDP | See <i>user datagram protocol (UDP)</i> . |
| unicode transformation format (UTF) | UTF is an industry standard that is intended to support the characters used by a large number of the world's languages. |
| uniform resource locator (URL) | The URL is the address of an Internet site. The URL contains the protocol used for the site—such as http or ftp, the domain name, or IP address of the site. For example, optionally, the folder or page on the site where specific information is stored. |
| URL | See <i>uniform resource locator (URL)</i> . |
| user datagram protocol (UDP) | UDP is a core protocol of the Internet protocol suite. |
| UTF | See <i>unicode transformation format (UTF)</i> . |
| VBI | See <i>vertical blanking interval (VBI)</i> . |
| vestigial sideband (VSB) | VSB is the modulation mode and can be specified to either an 8- or 16-level. |
| vertical blanking interval (VBI) | VBI is a portion of the television signal that does not contain visual data. In NTSC, the VBI are lines 1 through 21 in each field. |
| VOD | See <i>video on demand (VOD)</i> . |
| video on demand (VOD) | VOD allows a subscriber to select movies to view from a large selection of titles and categories stored on a remote server at any time. This service may also provide VCR functionality—stop, pause, etc.—that allows the subscriber to control the play back of the server from the remote control. |
| VSB | See <i>vestigial sideband (VSB)</i> . |
| XAIT | See <i>extended application information table (XAIT)</i> . |
| extensible markup language (XML) | XML is used to exchange of a wide variety of data. |

XML See *extensible markup language (XML)*.

XOR See *exclusive or (XOR)*.

Index

Symbols

%OCAPROOT% (parent) directory 4-2
_symbolDesc MOD-4

A

ActivePerl
 Installation 2-7
 Overview 4-9
ait-xxx.properties
 Applications 3-48
 Optional fields 3-50
 Required fields 3-48
 Example file 3-54
 External authorization 3-53
 Overview 3-46
 Transport protocols 3-46
 Version 3-46
Ant 4-8
apps directory 4-3
ARES 5-46
assets directory 4-3

B

bin directory 4-3
Build construction tools 4-9
 ActivePerl 4-9
 Cygwin 4-9
 Target-specific SDKs 4-10
Build control tools 4-8
 Ant 4-8
 omake 4-9
Build environment 4-1
Build files 4-11

Creating third-party OCAP extensions 4-24
Directory structure 4-2
Generating the build 4-15
Installing the OCAP stack on the host device 4-17
Jni3pExample OCAP extension 4-19
Legacy DVR recording extension 4-20
OCAP extensions 4-18
Software tools 4-8
Build files
 Overview 4-11

C

Class libraries
 OCAP 1-3
Closed captioning
 Data types and structures CAP-3
 Definitions CAP-2
 Overview CAP-1
 Supported functions CAP-13

Closed captioning functions
 mpeos_ccGetAnalogServices CAP-1
 4
 mpeos_ccGetAttributes CAP-15
 mpeos_ccGetCapability CAP-16
 mpeos_ccGetCaptioning CAP-18
 mpeos_ccGetDigitalServices CAP-1
 9
 mpeos_ccGetSupportedServiceNumbers CAP-20
 mpeos_ccSetAnalogServices CAP-2
 1

| | | | |
|---------------------------------|---------|--------------------------------------|---------|
| mpeos_ccSetAttributes | CAP-22 | mpeos_dbgStatusUnregister | DBG-1 |
| mpeos_ccSetClosedCaptioning | CAP-24 | | 7 |
| mpeos_ccSetDigitalServices | CAP-25 | mpeos_dbgStatusUnregisterInterest | DBG-18 |
| Common download | | Development system | |
| Data types and structures | CDL-3 | Configuring | 2-4 |
| Error codes | CDL-2 | Hardware requirements | 1-4 |
| Events | CDL-4 | Software requirements | 1-5 |
| Overview | CDL-1 | DirectFB | |
| Supported functions | CDL-5 | Directory structure | |
| Common download functions | | %OCAPROOT% variable | 4-2 |
| mpeos_cdlDownload | CDL-7 | apps | 4-3 |
| mpeos_cdlRegister | CDL-6 | assets | 4-3 |
| mpeos_cdlUnregister | CDL-8 | bin | 4-3 |
| Configuring | | Build environment | 4-2 |
| ait-xxx.properties file | 3-46 | docs | 4-3 |
| Development system | 2-4 | extensions | 4-4 |
| hostapp.properties file | 3-65 | java | 4-4 |
| Monitor application | 3-75 | jni | 4-4 |
| mpeenv.ini file | 3-3 | jvm | 4-4 |
| SDK | 2-11 | mpe | 4-5 |
| xait.properties file | 3-55 | other | 4-5 |
| Your environment | 3-1 | target | 4-6 |
| Contents of package | | tools | 4-6 |
| Full source | 1-6 | Top-level directories | 4-2 |
| Porting kit | 1-6 | Display | |
| Conventions | | Background and video reference model | DSP-2 |
| Coordinate system | | Data types and structures | DSP-7 |
| Cygwin | | Error codes | DSP-6 |
| Overview | 4-9 | Events | DSP-20 |
| D | | Overview | DSP-1 |
| dbg_logViaUDP | | Supported functions | DSP-21 |
| Debug | | Display functions | |
| Data types and structures | DBG-6 | mpeos_dispBGImageDelete | DSP-23 |
| Definitions | DBG-4 | mpeos_dispBGImageGetSize | DSP-24 |
| Error codes | DBG-5 | mpeos_dispBGImageNew | DSP-25 |
| Java-level logging | DBG-2 | mpeos_dispCheckDFC | DSP-26 |
| Logging commands | DBG-3 | mpeos_dispDisplayBGImage | DSP-28 |
| Logging examples | DBG-3 | mpeos_dispEnableOutputPort | DSP-29 |
| Logging overview | DBG-2 | mpeos_dispFlushGfxSurface | DSP-30 |
| Native C level logging | DBG-2 | mpeos_dispGetBGCColor | DSP-31 |
| Overview | DBG-1 | mpeos_dispGetCoherentConfigCount | DSP-32 |
| Supported functions | DBG-11 | mpeos_dispGetCoherentConfigs | DS-P-33 |
| Debug functions | | mpeos_dispGetConfigCount | DSP-34 |
| dbg_logViaUDP | DBG-12 | mpeos_dispGetConfigInfo | DSP-35 |
| mpeos_dbgLogUDP | DBG-13 | | |
| mpeos_dbgStatusGetTypes | DBG-14 | | |
| mpeos_dbgStatusRegister | DBG-15 | | |
| mpeos_dbgStatusRegisterInterest | D-BG-16 | | |

mpeos_dispGetConfigs DSP-36
 mpeos_dispGetConfigSet DSP-37
 mpeos_dispGetConfigSetCount DS
 P-38
 mpeos_dispGetCurrConfig DSP-39
 mpeos_dispGetDeviceCount DSP-4
 0
 mpeos_dispGetDeviceInfo DSP-41
 mpeos_dispGetDevices DSP-42
 mpeos_dispGetDFC DSP-43
 mpeos_dispGetGfxSurface DSP-45
 mpeos_dispGetOutputPortCount D
 SP-46
 mpeos_dispGetOutputPortInfo DSP
 -47
 mpeos_dispGetOutputPorts DSP-48
 mpeos_dispGetRFBypassState DSP-
 49
 mpeos_dispGetRFChannel DSP-50
 mpeos_dispGetScreenCount DSP-5
 1
 mpeos_dispGetScreenInfo DSP-52
 mpeos_dispGetScreens DSP-53
 mpeos_dispGetVideoOutputPortOpti
 on DSP-54
 mpeos_dispSetBGColor DSP-56
 mpeos_dispSetCoherentConfig DSP
 -57
 mpeos_dispSetCurrConfig DSP-58
 mpeos_dispSetDFC DSP-59
 mpeos_dispSetRFBypassState DSP-
 61
 mpeos_dispSetRFChannel DSP-62
 mpeos_dispSetVideoOutputPortOpti
 on DSP-63
 mpeos_dispWouldImpact DSP-65
 docs directory 4-3
 Documentation overview ii-xxxviii
 DVR
 Data types and structures DVR-3
 Definitions DVR-2
 Events DVR-14
 Legacy DVR recording
 extension 4-20
 Overview DVR-1
 Source files 5-4
 Supported functions DVR-16
 DVR functions
 mpeos_dvrFreeRecordingList DVR-1
 9
 mpeos_dvrGet DVR-20
 mpeos_dvrGetLowPowerResumeTim
 e DVR-21
 mpeos_dvrGetPlayScales DVR-22
 mpeos_dvrGetRecordingList DVR-2
 3
 mpeos_dvrGetSystemStatus DVR-2
 4
 mpeos_dvrGetTrickMode DVR-25
 mpeos_dvrIsDecodable DVR-26
 mpeos_dvrIsDecryptable DVR-27
 mpeos_dvrMediaVolumeAddAlarm
 DVR-28
 mpeos_dvrMediaVolumeDelete DV
 R-29
 mpeos_dvrMediaVolumeGetCount
 DVR-30
 mpeos_dvrMediaVolumeGetInfo D
 VR-31
 mpeos_dvrMediaVolumeGetList DV
 R-32
 mpeos_dvrMediaVolumeNew DVR-
 33
 mpeos_dvrMediaVolumeRegisterQue
 ue DVR-35
 mpeos_dvrMediaVolumeRemoveAlar
 m DVR-36
 mpeos_dvrMediaVolumeSetInfo DV
 R-37
 mpeos_dvrPlaybackChangePids DV
 R-39
 mpeos_dvrPlaybackGetPids DVR-40
 mpeos_dvrPlaybackGetTime DVR-4
 1
 mpeos_dvrPlaybackSetTime DVR-4
 2
 mpeos_dvrPlaybackStop DVR-43
 mpeos_dvrRecordingDelete DVR-44
 mpeos_dvrRecordingGet DVR-45
 mpeos_dvrRecordingPlayStart DVR-
 47
 mpeos_dvrResumeFromLowPower
 DVR-49
 mpeos_dvrSetTrickMode DVR-50
 mpeos_dvrTsbBufferingChangePids
 DVR-51
 mpeos_dvrTsbBufferingStart DVR-
 52
 mpeos_dvrTsbBufferingStop DVR-5
 4
 mpeos_dvrTsbChangeDuration DV
 R-55
 mpeos_dvrTsbConvertChangePids
 DVR-56
 mpeos_dvrTsbConvertStart DVR-5
 7
 mpeos_dvrTsbConvertStop DVR-5
 9

| | | | |
|--------------------------------------|--------|---|--------|
| mpeos_dvrTsbDelete | DVR-60 | mpeos_persistentFileInfoDelete | PER-8 |
| mpeos_dvrTsbGet | DVR-61 | mpeos_persistentGetStatFileName | PER-6 |
| mpeos_dvrTsbNew | DVR-62 | mpeos_persistentGetTempStatFileName | PER-7 |
| mpeos_dvrTsbPlayStart | DVR-63 | mpeos_persistentSetDefaultFileAttributes | PER-10 |
| E | | | |
| Environment | | | |
| ait-xxx.properties | 3-46 | File persistent | |
| hostapp.properties | 3-65 | Error codes | PER-2 |
| monitor application | 3-75 | File system | |
| mpeenv.ini | 3-3 | Broadcast data types and structures | FIL-21 |
| Overview | 3-1 | Broadcast file system file-status definitions | FIL-5 |
| Signaling files | 3-45 | Data types and structures | FIL-16 |
| xait.properties | 3-55 | Definitions | FIL-3 |
| Ethernet cable | 1-4 | Error codes | FIL-13 |
| Event | | File status definitions | FIL-3 |
| Data types and structures | EVT-3 | General functions | FIL-28 |
| Error codes | EVT-2 | Normal data types and structures | FIL-25 |
| Overview | EVT-1 | Operating-system specific definitions | FIL-7 |
| Supported functions | EVT-4 | Overview | FIL-1 |
| Test code | 6-5 | Permission definitions | FIL-8 |
| Event functions | | Porting note | FIL-2 |
| mpeos_eventQueueDelete | EVT-5 | Seek definitions | FIL-11 |
| mpeos_eventQueueNew | EVT-6 | Source files | 5-5 |
| mpeos_eventQueueNext | EVT-7 | Specific functions | FIL-31 |
| mpeos_eventQueueSend | EVT-8 | Type-support definitions | FIL-12 |
| mpeos_eventQueueWaitNext | EVT-9 | File system functions | |
| Events | | mpeos_dirClose | FIL-33 |
| Common download | CDL-4 | mpeos_dirCreate | FIL-34 |
| Display | DSP-20 | mpeos_dirDelete | FIL-35 |
| DVR | DVR-14 | mpeos_dirGetUStat | FIL-36 |
| Media | MED-17 | mpeos_dirMount | FIL-37 |
| POD | POD-7 | mpeos_dirOpen | FIL-38 |
| Section filtering | FLT-11 | mpeos_dirRead | FIL-39 |
| VBI | VBI-11 | mpeos_dirRename | FIL-40 |
| Extensions | | mpeos_dirSetUStat | FIL-41 |
| Creating third-party OCAP extensions | 4-24 | mpeos_dirUnmount | FIL-42 |
| Jni3pExample | 4-19 | mpeos_fileClose | FIL-43 |
| Legacy DVR recording | 4-20 | mpeos_fileDelete | FIL-44 |
| OCAP | 4-18 | mpeos_fileGetFStat | FIL-45 |
| extensions directory | 4-4 | mpeos_fileGetStat | FIL-47 |
| F | | | |
| File persistent | | mpeos_fileOpen | FIL-49 |
| Data types and structures | PER-3 | mpeos_fileRead | FIL-51 |
| Overview | PER-1 | mpeos_fileRemoveChangeListener | FIL-53 |
| Supported functions | PER-5 | mpeos_fileRename | FIL-52 |
| File persistent functions | | mpeos_fileSetChangeListener | FIL-54 |
| mpeos_fileNtfsDirGetUStat | PER-9 | | |

mpeos_fileSetFStat FIL-55
 mpeos_fileSetStat FIL-57
 mpeos_fileSync FIL-59
 mpeos_filesysGetDefaultDir FIL-29
 mpeos_filesysInit FIL-30
 mpeos_fileWrite FIL-60
 mpeos_init FIL-61
File systems
 Open attribute definitions FIL-6
 Priority definitions FIL-10
Fonts DSP-5
 Installing resident fonts 2-9
FreeType2 5-46
Front panel
 Data types and structures FPL-6
 Error codes FPL-5
 Overview FPL-1
 Supported functions FPL-9
 Test code 6-5
Front panel functions
 mpeos_fpGetCapabilities FPL-10
 mpeos_fpGetIndicator FPL-11
 mpeos_fpGetText FPL-13
 mpeos_fpInit FPL-15
 mpeos_fpSetIndicator FPL-16
 mpeos_fpSetText FPL-18
Full source
 Contents of package 1-6
 Installation 2-2

G

GFX_LOCK GFX-8
GFX_UNLOCK GFX-8
Graphic
 Source files 5-5
Graphic context DSP-4
Graphic manager
 Context data types and
 structures GFX-24
 Context functions GFX-34
 ConvertUTF.h GFX-3
 Coordinate system GFX-6
 Directory structure GFX-3
 Drawing functions GFX-52
 Font data types and
 structures GFX-25
 Font factory data types and
 structures GFX-27
 Font functions GFX-75
 Font-factory functions GFX-86
 Fonts GFX-7
 General data types and

 structures GFX-10
 General definitions GFX-8
 Graphic context GFX-6
 mpeos_context.h GFX-3
 mpeos_draw.h GFX-3
 mpeos_font.h GFX-3
 mpeos_fontfact.h GFX-3
 mpeos_gfx.h GFX-3
 mpeos_screen.h GFX-3
 mpeos_surface.h GFX-3
 mpeos_uievent.h GFX-3
 Overview GFX-1
 Reference model GFX-4
 Screen data types and
 structures GFX-28
 Screen functions GFX-90
 Screen layout GFX-4
 Supported data types and
 structures GFX-10
 Supported functions GFX-34
 Surface GFX-6
 Surface data types and
 structures GFX-30
 Surface functions GFX-94
 User-interface functions GFX-106

Graphic manager functions

mpeos_gfxBitBlt GFX-54
 mpeos_gfxClearRect GFX-55
 mpeos_gfxContextCreate GFX-36
 mpeos_gfxContextDelete GFX-37
 mpeos_gfxContextNew GFX-38
 mpeos_gfxCreateDefaultScreen GF
 X-91
 mpeos_gfxDrawArc GFX-56
 mpeos_gfxDrawEllipse GFX-57
 mpeos_gfxDrawLine GFX-58
 mpeos_gfxDrawPolygon GFX-59
 mpeos_gfxDrawPolyline GFX-60
 mpeos_gfxDrawRect GFX-61
 mpeos_gfxDrawRoundRect GFX-62
 mpeos_gfxDrawString GFX-64
 mpeos_gfxDrawString16 GFX-66
 mpeos_gfxFillArc GFX-68
 mpeos_gfxFillEllipse GFX-70
 mpeos_gfxFillPolygon GFX-71
 mpeos_gfxFillRect GFX-72
 mpeos_gfxFillRoundRect GFX-73
 mpeos_gfxFontDelete GFX-76
 mpeos_gfxFontFactoryAdd GFX-87
 mpeos_gfxFontFactoryDelete GFX-
 88
 mpeos_gfxFontFactoryNew GFX-8

| | |
|--|--|
| <p style="text-align: center;">9</p> <p>mpeos_gfxFontGetList GFX-77 mpeos_gfxFontHasCode GFX-78 mpeos_gfxFontNew GFX-79 mpeos_gfxGetCharWidth GFX-83 mpeos_gfxGetClipRect GFX-39 mpeos_gfxGetColor GFX-40 mpeos_gfxGetFont GFX-41 mpeos_gfxGetFontMetrics GFX-81 mpeos_gfxGetOrigin GFX-42 mpeos_gfxGetPaintMode GFX-43 mpeos_gfxGetScreen GFX-92 mpeos_gfxGetString16Width GFX-8 5 mpeos_gfxGetStringWidth GFX-84 mpeos_gfxGetSurface GFX-45 mpeos_gfxPaletteDelete GFX-95 mpeos_gfxPaletteGet GFX-96 mpeos_gfxPaletteMatch GFX-97 mpeos_gfxPaletteNew GFX-98 mpeos_gfxPaletteSet GFX-99 mpeos_gfxScreenResize GFX-93 mpeos_gfxSetClipRect GFX-46 mpeos_gfxSetColor GFX-47 mpeos_gfxSetFont GFX-48 mpeos_gfxSetOrigin GFX-49 mpeos_gfxSetPaintMode GFX-50 mpeos_gfxStretchBlt GFX-74 mpeos_gfxSurface GFX-100 mpeos_gfxSurfaceCreate GFX-101 mpeos_gfxSurfaceDelete GFX-102 mpeos_gfxSurfaceGetInfo GFX-103 mpeos_gfxSurfaceNew GFX-105 mpeos_gfxWaitNextEvent GFX-10 7</p> <p style="text-align: center;">H</p> <p>Hardware overview ii-xxxvii Hardware requirements 1-4 Development system 1-4 Ethernet cable 1-4 Host device 1-4 Serial cable 1-4 Television 1-4 Host device 1-3 Hardware requirements 1-4 Installing the OCAP stack 4-17 Software requirements 1-5 hostapp.properties Applications 3-67 Optional fields 3-70 Required fields 3-68</p> | <p>Example file 3-74 Overview 3-65 Services 3-73 Optional fields 3-74 Required fields 3-73 Transport protocols 3-66 Version 3-65</p> <p>Host-device applications Organization and design 5-44 Overview 5-43 Provisioning 5-43</p> <p style="text-align: center;">I</p> <p>Implementation Overview 5-3</p> <p>Installing ActivePerl 2-7 Full source 2-2 Java 2-5 Java2 SDK v1.3.1_07 2-5 Java2 SDK v1.4.2 2-5 OCAP stack on the host device 4-17 Porting kit 2-2 Resident fonts 2-9 SDK 2-8 Software overview 2-1</p> <p style="text-align: center;">J</p> <p>Java Classes 1-2 Installation 2-5 java directory 4-4 Java2 SDK v1.3.1_07 Installation 2-5 Java2 SDK v1.4.2 Installation 2-5 JNI 1-3 Porting layer 1-3 jni directory 4-4 Jni3pExample OCAP extension 4-19 JVM Launching 5-47 Porting layer 1-2 jvm directory 4-4</p> <p style="text-align: center;">L</p> <p>Launching the JVM 5-47 Legacy DVR recording extension 4-20</p> <p style="text-align: center;">M</p> <p>main.c 6-2</p> |
|--|--|

Makefile 6-2

Media

- Data types and structures MED-6
- Error codes MED-2
- Events MED-7
- Overview MED-1
- Source files 5-5
- Supported functions MED-22

Media functions

- mpeos_mediaCheckBounds MED-2
 4
- mpeos_mediaDecode MED-25
- mpeos_mediaDripFeedRenderFrame MED-27
- mpeos_mediaDripFeedStart MED-2
 8
- mpeos_mediaDripFeedStop MED-2
 9
- mpeos_mediaFreeze MED-30
- mpeos_mediaFrequencyToTuner MED-31
- mpeos_mediaGetAFD MED-32
- mpeos_mediaGetAspectRatio MED-34
- mpeos_mediaGetBounds MED-35
- mpeos_mediaGetInputVideoSize MED-36
- mpeos_mediaGetScaling MED-37
- mpeos_mediaGetTunerInfo MED-4
 0
- mpeos_mediaInit MED-41
- mpeos_mediaRegisterQueueForTune
 Events MED-42
- mpeos_mediaResume MED-43
- mpeos_mediaSetBounds MED-44
- mpeos_mediaShutdown MED-45
- mpeos_mediaStop MED-46
- mpeos_mediaSwapDecoders MED-47
- mpeos_mediaTune MED-48
- mpeos_mediaUnregisterQueue MED-50

Memory

- Allocation failure handling MEM-2
- Data types and structures MEM-7
- Definitions MEM-4
- Error codes MEM-6
- Overview MEM-1
- Supported functions MEM-11
- Test code 6-6

Memory functions

- mpeos_memAllocH MEM-13
- mpeos_memAllocPGen MEM-14
- mpeos_memAllocPProf MEM-15
- mpeos_memCompact MEM-16
- mpeos_memFreeH MEM-17
- mpeos_memFreePGen MEM-18
- mpeos_memFreePProf MEM-19
- mpeos_memGetFreeSize MEM-20
- mpeos_memGetLargestFree MEM-21
- mpeos_memGetStats MEM-22
- mpeos_memInit MEM-23
- mpeos_memLockH MEM-24
- mpeos_memPurge MEM-25
- mpeos_memReallocH MEM-26
- mpeos_memReallocPGen MEM-27
- mpeos_memReallocPProf MEM-28
- mpeos_memRegisterMemFreeCallback MEM-29
- mpeos_memStats MEM-31
- mpeos_memUnregisterMemFreeCallback MEM-32

Miscellaneous

- Source files 5-6

Modifying

- Test code 6-4

Module support

- Data types and structures MOD-3
- Error codes MOD-2
- Overview MOD-1
- Supported functions MOD-5

Module support functions

- mpeos_dlmodClose MOD-6
- mpeos_dlmodGetSymbol MOD-7
- mpeos_dlmodInit MOD-8
- mpeos_dlmodOpen MOD-9

Monitor application

- Overview 3-75

Monitor application

- Configuring
 - hostapp.properties file 3-76
 - mpeenv.ini file 3-77
 - Signaling the monitor application 3-75

MPE

- APIs I-3
- Service managers I-3

mpe directory 4-5

MPE_ACTIVE_FORMAT_CHANGED MED-17

MPE_ASPECT_RATIO_CHANGED MED-18

mpe_Bool DBG-6, DSP-7, DVR-3,
 FIL-16, GFX-10, MED-6,
 MEM-7, NET-23, POD-3,
 SND-3, STG-3, SYN-3, UTL-4
 MPE_BS_MPE_LOWLVL UTL-2
 MPE_BS_MPE_MGRS UTL-2
 MPE_BS_NET_IWAY UTL-2
 MPE_BS_NET_2WAY UTL-2
 MPE_BS_NET_MASK UTL-2
 MPE_BS_NET_NOWAY UTL-2
 MPE_CC_COLOR CAP-2
 MPE_CC_EMBEDDED_COLOR CAP-2
 mpe_CcAnalogServiceMap CAP-3
 mpe_CcAnalogServices CAP-3
 mpe_CcAttribType CAP-4
 mpe_CcAttributes CAP-5
 mpe_CcBorderType CAP-6
 mpe_CcColor CAP-7
 mpe_CcDigitalServiceMap CAP-7
 mpe_CcDigitalServices CAP-8
 mpe_CcError CAP-8
 mpe_CcFontSize CAP-8
 mpe_CcFontStyle CAP-9
 mpe_CcOpacity CAP-10
 mpe_CcState CAP-11
 mpe_CcTextStyle CAP-11
 mpe_CcType CAP-12
 mpe_Cond SYN-3
 MPE_CONTENT_PRESENTING ME
 D-18
 mpe_DbglStatusFormat DBG-6
 mpe_DbglStatusId DBG-6
 mpe_DbglStatusType DBG-6
 MPE_DFC_CHANGED DSP-20,
 MED-18
 mpe_Dir FIL-16
 mpe_DirEntry FIL-25
 mpe_DirInfo FIL-21
 mpe_DirStatMode FIL-21
 mpe_DirUrl FIL-21
 mpe_DisplBGImage DSP-8
 mpe_DisplCoherentConfig DSP-8
 mpe_DisplDevice DSP-8, DVR-3, MED-6
 mpe_DisplDeviceConfig DSP-9
 mpe_DisplDeviceConfigInfo DSP-9
 mpe_DisplDeviceInfo DSP-10
 mpe_DisplDeviceType DSP-11
 mpe_DisplDfcAction DSP-11
 mpe_DisplError DSP-12
 mpe_DisplI1394DeviceInfo DSP-7
 mpe_DisplI1394Devices DSP-8
 mpe_DisplOutputPort DSP-13
 mpe_DisplOutputPortInfo DSP-13
 mpe_DisplOutputPortOption DSP-14
 mpe_DisplOutputPortOptionName DSP-15
 mpe_DisplOutputPortOptionValue DSP-16
 mpe_DisplOutputPortType DSP-16
 mpe_DisplScreen DSP-17
 mpe_DisplScreenArea DSP-17
 mpe_DisplScreenInfo DSP-18
 mpe_Dlmod MOD-3
 mpe_DlmodData MOD-3
 MPE_DVR_EVT_DEVICE_ERROR DV
 R-14
 MPE_DVR_EVT_SYSTEM_BUSY DVR
 -I4
 MPE_DVR_EVT_SYSTEM_READY D
 VR-14
 MPE_DVR_MAX_NAME_SIZE DVR-2
 MPE_DVR_MAX_PIDS DVR-2
 MPE_DVR_MEDIA_VOL_MAX_PATH_
 SIZE DVR-2
 MPE_DVR_PID_UNKNOWN DVR-2
 MPE_DVR_POSITIVE_INFINITY DVR
 -2
 mpe_DvrBitRate DVR-3
 mpe_DvrBuffering DVR-3
 mpe_DvrConversion DVR-4
 mpe_DvrError DVR-4
 mpe_DvrEvent DVR-14
 mpe_DvrInfoParam DVR-5
 mpe_DvrMediaStreamType DVR-7
 mpe_DvrPidInfo DVR-8
 mpe_DvrPidTable DVR-8
 mpe_DvrPlayback DVR-8, VBI-5
 mpe_DvrState DVR-9
 mpe_DvrString_t DVR-9
 mpe_DvrTsb DVR-9
 MPE_EBUSY SYN-2
 MPE_ECOND SYN-2
 mpe_EdHandle SND-3
 MPE_EEVENT POD-2
 MPE_EINVAL FPL-5, MED-2, MEM-6,
 MOD-2, NET-17, POD-2,
 FLT-4, SND-2, SYN-2, THR-5,
 TME-2, UTL-3, VBI-3
 MPE_EMUTEX SYN-2
 MPE_ENODATA MEM-6, MOD-2,
 NET-17, POD-2
 MPE_ENOMEM MED-2, MEM-6,
 NET-17, POD-2, FLT-4, SND-2,
 SYN-2, THR-5, UTL-3, VBI-3

| | | |
|-----------------------------|---|--|
| mpe_Error | DVR-9, EVT-3, GFX-10, MED-6, MEM-7, MOD-3, POD-3, FLT-6, SYN-3, THR-6, TME-3, UTL-4, VBI-5 | MPE_FS_ERROR_DEVICE_FAILURE FIL-13 |
| MPE_ETHREADDEATH | NET-17 | MPE_FS_ERROR_EOF FIL-13 |
| mpe_Event | EVT-3 | MPE_FS_ERROR_EVENTS FIL-13 |
| MPE_EVENT_SHUTDOWN | MED-17 | MPE_FS_ERROR_FAILURE FIL-13 |
| mpe_EventQueue | DVR-9, EVT-3, FIL-22, MED-6, POD-3, FLT-6, STG-3, UTL-4, VBI-5 | MPE_FS_ERROR_INVALID_PARAMET ER FIL-13, PER-2 |
| MPE_FAILURE_CA_DENIED | MED-18 | MPE_FS_ERROR_INVALID_STATE F IL-14 |
| MPE_FAILURE_NO_DATA | MED-18 | MPE_FS_ERROR_NO_MOUNT FIL-14 |
| MPE_FAILURE_UNKNOWN | MED-18 | MPE_FS_ERROR_NOT_FOUND FIL-I 4 |
| mpe_File | FIL-16 | MPE_FS_ERROR NOTHING_TO_ABO RT FIL-14 |
| mpe_FileChangeHandle | FIL-16 | MPE_FS_ERROR_OUT_OF_MEM FIL -14, PER-2 |
| mpe_FileError | FIL-16, PER-3 | MPE_FS_ERROR_READ_ONLY FIL-14 |
| mpe_FileInfo | FIL-17 | MPE_FS_ERROR_SUCCESS FIL-14, PER-2 |
| mpe_FileOpenMode | FIL-19 | MPE_FS_ERROR_TIMEOUT FIL-14 |
| mpe_FileSeekMode | FIL-19 | MPE_FS_ERROR_UNKNOWN_URL F IL-15 |
| mpe_FileStatMode | FIL-19 | MPE_FS_ERROR_UNSUPPORT FIL-I 5 |
| mpe_FilterComponent | FLT-6, VBI-5 | MPE_FS_MAX_PATH FIL-7 |
| mpe_FilterSectionHandle | FLT-6 | MPE_FS_OPEN_APPEND FIL-6 |
| mpe_FilterSource | FLT-7 | MPE_FS_OPEN_CAN_CREATE FIL-6 |
| mpe_FilterSource_INB | FLT-8 | MPE_FS_OPEN_MUST_CREATE FIL- 6 |
| mpe_FilterSource_OOB | FLT-8 | MPE_FS_OPEN_READ FIL-6 |
| mpe_FilterSourceParams | FLT-9 | MPE_FS_OPEN_READWRITE FIL-6 |
| mpe_FilterSourceType | FLT-9 | MPE_FS_OPEN_TRUNCATE FIL-6 |
| mpe_FilterSpec | FLT-10, VBI-6 | MPE_FS_OPEN_WRITE FIL-7 |
| MPE_FP_BRIGHTNESS_FULL | FPL-2 | MPE_FS_PERM_GROUP_EXEC FIL-8 |
| MPE_FP_BRIGHTNESS_OFF | FPL-2 | MPE_FS_PERM_GROUP_READ FIL-8 |
| MPE_FP_COLOR_BLUE | FPL-2 | MPE_FS_PERM_GROUP_WRITE FIL- 8 |
| MPE_FP_COLOR_GREEN | FPL-2 | MPE_FS_PERM_OWNER_EXEC FIL- 8 |
| MPE_FP_COLOR_ORANGE | FPL-2 | MPE_FS_PERM_OWNER_READ FIL- 8 |
| MPE_FP_COLOR_RED | FPL-3 | MPE_FS_PERM_OWNER_WRITE FIL -8 |
| MPE_FP_COLOR_UNSUPPORTED | F PL-3 | MPE_FS_PERM_WORLD_EXEC FIL-8 |
| MPE_FP_COLOR_YELLOW | FPL-3 | MPE_FS_PERM_WORLD_READ FIL-9 |
| MPE_FP_INDICATOR_MESSAGE | FP L-3 | MPE_FS_PERM_WORLD_WRITE FIL -9 |
| MPE_FP_INDICATOR_POWER | FPL-3 | MPE_FS_PRIOR_HI FIL-10 |
| MPE_FP_INDICATOR_RECORD | FPL- 3 | MPE_FS_PRIOR_LOW FIL-10 |
| MPE_FP_INDICATOR_REMOTE | FPL -3 | MPE_FS_PRIOR_MED FIL-10 |
| MPE_FP_INDICATOR_RFBYPASS | FP L-3 | MPE_FS_SEEK_CUR FIL-II |
| MPE_FP_INDICATOR_TEXT | FPL-4 | MPE_FS_SEEK_END FIL-II |
| mpe_FpBlinkSpec | FPL-6 | MPE_FS_SEEK_SET FIL-II |
| mpe_FpCapabilities | FPL-6 | |
| mpe_FpScrollSpec | FPL-7 | |
| mpe_FpTextPanelMode | FPL-8 | |
| MPE_FS_ERROR_ALREADY_EXISTS | FIL-13 | |

MPE_FS_STAT_CONNECTIONAVAIL
 FIL-5
 MPE_FS_STAT_CREATEDATE FIL-3
 MPE_FS_STAT_EXPDATE FIL-3
 MPE_FS_STAT_ISKNOWN FIL-3
 MPE_FS_STAT_MODDATE FIL-4
 MPE_FS_STAT_MOUNTPATH FIL-5
 MPE_FS_STAT_ORG_ACCESS FIL-4
 MPE_FS_STAT_PERM FIL-4
 MPE_FS_STAT_PRIOR FIL-4
 MPE_FS_STAT_SIZE FIL-4
 MPE_FS_STAT_SOURCEID FIL-5
 MPE_FS_STAT_TYPE FIL-4
 MPE_FS_TYPE_DIR FIL-12
 MPE_FS_TYPE_FILE FIL-12
 MPE_FS_TYPE_OTHER FIL-12
 MPE_FS_TYPE_STREAM FIL-12
 MPE_FS_TYPE_STRAEVEVENT FIL-12
 MPE_FS_TYPE_UNKNOWN FIL-12
 MPE_GFX_UNKNOWN GFX-8
 MPE_GFX_WAIT_INFINITE GFX-8
 mpe_GfxAlphaChannel GFX-11
 mpe_gfxArgbToColor GFX-8
 mpe_GfxBitDepth GFX-12
 mpe_GfxColor DSP-18, GFX-12
 mpe_GfxColorFormat GFX-13
 mpe_GfxContext GFX-14
 mpe_GfxDimensions DSP-18, GFX-14,
 MED-6
 mpe_GfxError GFX-14
 mpe_GfxEvent GFX-15
 mpe_GfxFont GFX-16
 mpe_GfxFontDesc GFX-17
 mpe_GfxFontFactory GFX-18
 mpe_GfxFontFormat GFX-18
 mpe_GfxFontMetrics GFX-18
 mpe_GfxFontStyle GFX-19
 mpe_gfxGetAlpha GFX-9
 mpe_gfxGetBlue GFX-9
 mpe_gfxGetGreen GFX-9
 mpe_gfxGetRed GFX-9
 mpe_GfxPaintMode GFX-19
 mpe_GfxPalette GFX-20
 mpe_GfxPoint GFX-21
 mpe_GfxRectangle DSP-19, GFX-21
 mpe_gfxRgbToColor GFX-9
 mpe_GfxScreen GFX-21
 mpe_GfxSurface DSP-19, GFX-21
 mpe_GfxSurfaceInfo GFX-21
 mpe_GfxWchar GFX-23
 mpe_JmpBuf UTL-4
 mpe_logModule DBG-7
 mpe_MediaActiveFormatDescription M
 ED-7
 mpe_MediaAspectRatio MED-8
 mpe_MediaDecodeRequestParams MED
 -9
 mpe_MediaDecodeSession MED-9,
 VBI-6
 mpe_MediaDripFeedRequestParams ME
 D-9
 mpe_MediaErrorCode MED-3
 mpe_MediaPid MED-10
 mpe_MediaPositioningCapabilities MED-10
 mpe_MediaRectangle MED-11
 mpe_MediaTuneParams MED-11
 mpe_MediaTuneRequestParams MED-11
 mpe_MediaTuneType MED-12
 mpe_MediaVolume DVR-9
 mpe_MediaVolumeAllowedList DVR-9
 mpe_MediaVolumeEvent DVR-10
 mpe_MediaVolumeInfoParam DVR-10
 MPE_MEM_NOPURGE MEM-4
 MPE_MEM_PRIOR_HIGH MEM-4
 MPE_MEM_PRIOR_LOW MEM-4
 mpe_MemColor MEM-7
 mpe_MemHandle MEM-9
 mpe_MemStatsInfo MEM-9
 mpe_Mutex POD-3, SYN-3
 MPE POD_EVENT_APPINFO_UPDAT
E POD-7
 MPE POD_EVENT_GF_UPDATE PO
D-7
 mpe_PODAppInfo POD-3
 mpe_PODDatabase POD-4
 mpe_PODFeatureParams POD-5
 mpe_PODFeatures POD-4
 mpe_PowerStatus UTL-5
 MPE_SF_ERROR FLT-4
 MPE_SF_ERROR_FILTER_NOT_AVAI
LABLE FLT-4
 MPE_SF_ERROR_INVALID_SECTION_
HANDLE FLT-4
 MPE_SF_ERROR_SECTION_NOT_AV
AILABLE FLT-4
 MPE_SF_ERROR_TUNER_NOT_AT_F
REQUENCY FLT-5
 MPE_SF_ERROR_TUNER_NOT_TUNE
D FLT-5
 MPE_SF_EVENT_FILTER_AVAILABLE
FLT-13
 MPE_SF_EVENT_FILTER_CANCELLE
D FLT-13

| | | | |
|-------------------------------------|-------------------|-----------------------------|-------------|
| MPE_SF_EVENT_FILTER_PREEMPTED | FLT-13 | MPE_SOCKET_EHOSTNOTFOUND | NET-19 |
| MPE_SF_EVENT_LAST_SECTION_FOUND | UND | MPE_SOCKET_EHOSTUNREACH | N |
| MPE_SF_EVENT_OUT_OF_MEMORY | FLT-13 | MPE_SOCKET_EINTR | NET-19 |
| MPE_SF_EVENT_SECTION_FOUND | FLT-14 | MPE_SOCKET_EIO | NET-19 |
| MPE_SF_EVENT_SOURCE_CLOSED | FLT-14 | MPE_SOCKET_EISCONN | NET-19 |
| MPE_SF_EVENT_UNKNOWN | FLT-I 4 | MPE_SOCKET_ELOOP | NET-20 |
| MPE_SF_INVALID_SECTION_HANDLE | FLT-3 | MPE_SOCKET_EMFILE | NET-20 |
| MPE_SF_INVALID_UNIQUE_ID | FLT- 3 | MPE_SOCKET_EMSGSIZE | NET-20 |
| MPE_SF_OPTION_IF_NOT_CANCELED | FLT-3 | MPE_SOCKET_ENAMETOOLONG | NET-20 |
| MPE_SF_OPTION_RELEASE_WHEN_COMPLETE | FLT-3 | MPE_SOCKET_ENETDOWN | NET-2 |
| mpe_SiElemStreamType | DVR-11, MED-12 | MPE_SOCKET_ENETUNREACH | NE |
| mpe_SiModulationMode | FIL-22, MED-14 | T-20 | |
| mpe_SiServiceHandle | FIL-20 | MPE_SOCKET_ENFILE | NET-20 |
| mpe_SndDevice | SND-3 | MPE_SOCKET_ENOBUFS | NET-21 |
| mpe_SndError | SND-3 | MPE_SOCKET_ENOPROTOOPT | NE |
| mpe_SndEvent | SND-4 | T-21 | |
| mpe_SndPlayback | SND-4 | MPE_SOCKET_ENORECOVERY | NE |
| mpe_SndSound | SND-4 | T-21 | |
| mpe_Socket | NET-23 | MPE_SOCKET_ENOSPC | NET-21 |
| MPE_SOCKET_AF_INET4 | NET-5 | MPE_SOCKET_ENOTCONN | NET-21 |
| MPE_SOCKET_AF_INET6 | NET-7 | MPE_SOCKET_ENOTSOCK | NET-21 |
| MPE_SOCKET_DATAGRAM | NET-2 | MPE_SOCKET_EOPNOTSUPP | NET-21 |
| MPE_SOCKET_EACCES | NET-18 | MPE_SOCKET_EPIPE | NET-21 |
| MPE_SOCKET_EADDRINUSE | NET-I 8 | MPE_SOCKET_EPROTO | NET-22 |
| MPE_SOCKET_EADDRNOTAVAIL | N | MPE_SOCKET_EPROTONOSUPPORT | NET-22 |
| ET-18 | | MPE_SOCKET_EPROTOTYPE | NET-22 |
| MPE_SOCKET_EAFNOSUPPORT | NE | MPE_SOCKET_ETIMEDOUT | NET-2 |
| T-18 | | MPE_SOCKET_ETRYAGAIN | NET-22 |
| MPE_SOCKET_EAGAIN | NET-18 | MPE_SOCKET_EWOULDBLOCK | NE |
| MPE_SOCKET_EALREADY | NET-18 | T-22 | |
| MPE_SOCKET_EBADF | NET-18 | MPE_SOCKET_FD_SETSIZE | NET-2 |
| MPE_SOCKET_ECONNABORTED | N | MPE_SOCKET_FIONBIO | NET-2 |
| ET-18 | | MPE_SOCKET_FIONREAD | NET-3 |
| MPE_SOCKET_ECONNREFUSED | N | MPE_SOCKET_IN4ADDR_ANY | NET-5 |
| ET-19 | | MPE_SOCKET_IN4ADDR_LOOPBACK | NET-5 |
| MPE_SOCKET_ECONNRESET | NET- 19 | MPE_SOCKET_IN6ADDR_ANY_INIT | NET-7 |
| MPE_SOCKET_EDESTADDRREQ | NE | MPE_SOCKET_IN6ADDR_LOOPBACK | _INIT NET-7 |
| T-19 | | MPE_SOCKET_INET4_ADDRSTRLEN | NET-5 |
| MPE_SOCKET_EDOM | NET-19 | MPE_SOCKET_INET6_ADDRSTRLEN | NET-7 |

| | | | |
|---------------------------------|------------|-----------------------------------|---------------------|
| MPE_SOCKET_INVALID_SOCKET | N ET-3 | MPE_SOCKET_SO_SNDBUF | NET-10 |
| MPE_SOCKET IPPROTO_IPV4 | NET-5 | MPE_SOCKET_SO SNDLOWAT | NE T-10 |
| MPE_SOCKET IPPROTO_IPV6 | NET-7 | MPE_SOCKET_SO SNDTIMEO | NET -10 |
| MPE_SOCKET IPPROTO_TCP | NET-5 | MPE_SOCKET_SO_TYPE | NET-11 |
| MPE_SOCKET IPPROTO_UDP | NET-6 | MPE_SOCKET_SOL_SOCKET | NET-1 I |
| MPE_SOCKET_IPV4_ADD_MEMBERSHIP | HIP NET-12 | MPE_SOCKET_STREAM | NET-4 |
| MPE_SOCKET_IPV4_DROP_MEMBERSHIP | NET-12 | MPE_SOCKET_TCP_NODELAY | NET -16 |
| MPE_SOCKET_IPV4_MULTICAST_IF | NET-13 | mpe_SocketFDSet | NET-23 |
| MPE_SOCKET_IPV4_MULTICAST_LOOP | NET-13 | mpe_SocketHostEntry | NET-23 |
| MPE_SOCKET_IPV4_MULTICAST_TTL | NET-13 | mpe_SocketIPv4Addr | NET-25 |
| MPE_SOCKET_IPV6_ADD_MEMBERSHIP | HIP NET-14 | mpe_SocketIPv4McastReq | NET-25 |
| MPE_SOCKET_IPV6_DROP_MEMBERSHIP | NET-14 | mpe_SocketIPv4SockAddr | NET-25 |
| MPE_SOCKET_IPV6_MULTICAST_HOPS | NET-15 | mpe_SocketIPv6Addr | NET-26 |
| MPE_SOCKET_IPV6_MULTICAST_IF | NET-15 | mpe_SocketIPv6McastReq | NET-26 |
| MPE_SOCKET_IPV6_MULTICAST_LOOP | NET-15 | mpe_SocketIPv6SockAddr | NET-26 |
| MPE_SOCKET_MAXHOSTNAMELEN | NET-3 | mpe_SocketLinger | NET-24 |
| MPE_SOCKET_MSG_OOB | NET-3 | mpe_SocketSaFamily | NET-24 |
| MPE_SOCKET_MSG_PEEK | NET-3 | mpe_SocketSockAddr | NET-24 |
| MPE_SOCKET_SHUT_RD | NET-3 | mpe_SocketSockLen | NET-24 |
| MPE_SOCKET_SHUT_RDWR | NET-3 | mpe_STBBootMode | UTL-4 |
| MPE_SOCKET_SHUT_WR | NET-4 | MPE_STILL_FRAME_DECODED | ME D-20 |
| MPE_SOCKET_SO_BROADCAST | NET-8 | mpe_Storage | DVR-13 |
| MPE_SOCKET_SO_DEBUG | NET-8 | MPE_STORAGE_MAX_DISPLAY_NAME_SIZE | NA ME_SIZE STG-2 |
| MPE_SOCKET_SO_DONTROUTE | NET-8 | MPE_STORAGE_MAX_NAME_SIZE | STG-2 |
| MPE_SOCKET_SO_ERROR | NET-8 | MPE_STORAGE_MAX_PATH_SIZE | S TG-2 |
| MPE_SOCKET_SO_KEEPALIVE | NET-9 | mpe_StorageError | STG-3 |
| MPE_SOCKET_SO_LINGER | NET-9 | mpe_StorageEvent | STG-9 |
| MPE_SOCKET_SO_OOBINLINE | NET-9 | mpe_StorageHandle | DVR-13, STG-4 |
| MPE_SOCKET_SO_RCVBUF | NET-9 | mpe_StorageInfoParam | STG-4 |
| MPE_SOCKET_SO_RCVLOWAT | NET-9 | mpe_StorageStatus | STG-6 |
| MPE_SOCKET_SO_RCVTIMEO | NET-10 | mpe_StorageSupportedAccessRights | ST G-7 |
| MPE_SOCKET_SO_REUSEADDR | NET-10 | mpe_Stream | FIL-24 |
| | | MPE_STREAM_CA_DENIED | MED-19 |
| | | MPE_STREAM_CA_UNKNOWN | ME D-19 |
| | | MPE_STREAM_DIALOG_PAYMENT | MED-19 |
| | | MPE_STREAM_DIALOG_RATING | M ED-19 |
| | | MPE_STREAM_DIALOG_TECHNICAL | MED-19 |
| | | MPE_STREAM_HW_UNAVAILABLE | MED-20 |
| | | MPE_STREAM_NO_DATA | MED-20 |

MPE_STREAM_RETURNED MED-20
 mpe_StreamEventInfo FIL-24
 MPE_SUCCESS FPL-5, MED-2,
 MEM-6, MOD-2, POD-2,
 FLT-5, SND-2, SYN-2, THR-5,
 TME-2, UTL-3, VBI-3
 MPE_THREAD_PRIOR_DFLT THR-3
 MPE_THREAD_PRIOR_INC THR-3
 MPE_THREAD_PRIOR_MAX THR-3
 MPE_THREAD_PRIOR_MIN THR-3
 MPE_THREAD_PRIOR_SYSTEM TH
 R-3
 MPE_THREAD_PRIOR_SYSTEM_HI
 THR-4
 MPE_THREAD_PRIOR_SYSTEM_MED
 THR-4
 MPE_THREAD_STACK_SIZE THR-4
 MPE_THREAD_STAT_CALLB THR-
 4
 MPE_THREAD_STAT_COND THR-4
 MPE_THREAD_STAT_DEATH THR-
 4
 MPE_THREAD_STAT_EVENT THR-
 4
 MPE_THREAD_STAT_MUTEX THR-
 4
 mpe_Threadid THR-6
 mpe_ThreadPrivateData THR-6
 mpe_ThreadStat THR-6
 mpe_Time FIL-20, PER-3, TME-3
 mpe_TimeClock TME-3
 mpe_TimeMillis TME-3
 mpe_TimeTm TME-3
 mpe_TimeVal NET-24, TME-3
 MPE_TUNE_ABORT MED-21
 MPE_TUNE_COMPLETE MED-21
 MPE_TUNE_FAIL MED-21
 MPE_TUNE_STARTED MED-21
 MPE_VBI_ERROR VBI-3
 MPE_VBI_ERROR_FILTER_NOT_AVAI
 LABLE VBI-3
 MPE_VBI_ERROR_INVALID_FILTER_S
 ESSION VBI-4
 MPE_VBI_ERROR_SOURCE_CLOSED
 VBI-4
 MPE_VBI_ERROR_SOURCE_SCRAMB
 LED VBI-4
 MPE_VBI_ERROR_UNSUPPORTED V
 BI-4
 MPE_VBI_EVENT_BUFFER_FULL VB
 I-II
 MPE_VBI_EVENT_DATAUNITS_REC
 EIVED VBI-II
 MPE_VBI_EVENT_FILTER_AVAILABL
 E VBI-12
 MPE_VBI_EVENT_FILTER_STOPPED
 VBI-12
 MPE_VBI_EVENT_FIRST_DATAUNIT
 VBI-12
 MPE_VBI_EVENT_OUT_OF_MEMORY
 Y VBI-12
 MPE_VBI_EVENT_SOURCE_CLOSED
 VBI-13
 MPE_VBI_EVENT_SOURCE_SCRAMB
 LED VBI-13
 MPE_VBI_EVENT_UNKNOWN VBI-1
 3
 MPE_VBI_OPTION_CLEAR_BUFFER
 VBI-2
 MPE_VBI_OPTION_CLEAR_READ_DA
 TA VBI-2
 MPE_VBI_OPTION_IF_NOT_STOPPE
 D VBI-2
 mpe_VBIDataFormat VBI-6
 mpe_VBIFields VBI-8
 mpe_VBIFilterSession VBI-8
 mpe_VBIParameter VBI-8
 mpe_VBISessionParameter VBI-9
 mpe_VBISource VBI-9
 mpe_VBISourceParams VBI-10
 mpe_VBISourceType VBI-10
 mpeenv.ini
 Configuring 3-3
 JVM-specific format 3-38
 JVM-specific overview 3-38
 JVM-specific properties and
 values 3-39
 OCAP stack-specific categories 3-22
 OCAP stack-specific fields
 DVR 3-22
 EAS 3-25
 Host 3-26
 Miscellaneous 3-22, 3-26
 MPE 3-27
 Object carousel 3-27
 Service information 3-32
 Signaling 3-37
 Time 3-38
 Port-specific categories 3-3
 Port-specific fields
 Debug 3-4
 Display 3-7
 File system 3-8
 Font 3-10
 Front panel 3-11
 Graphic 3-12
 Manager 3-12
 Memory 3-13

Network 3-17
 OCAP extensions 3-18
 Security 3-19
 System commands 3-20
 System host 3-20
 User interface 3-21

MPEOS

- Overview 5-1
- Porting layer 1-3
- Testing 6-1
- `mpeos_caption.c` 5-5
- `mpeos_ccGetAnalogServices` CAP-14
- `mpeos_ccGetAttributes` CAP-15
- `mpeos_ccGetCapability` CAP-16
- `mpeos_ccGetClosedCaptioning` CAP-18
- `mpeos_ccGetDigitalServices` CAP-19
- `mpeos_ccGetSupportedServiceNumbers` CAP-20
- `mpeos_ccSetAnalogServices` CAP-21
- `mpeos_ccSetAttributes` CAP-22
- `mpeos_ccSetClosedCaptioning` CAP-24
- `mpeos_ccSetDigitalServices` CAP-25
- `mpeos_cdl.c` 5-6
- `mpeos_cdlRegister` CDL-6
- `mpeos_cdlStartDownload` CDL-7
- `mpeos_cdlUnregister` CDL-8
- `mpeos_dbg.c` 5-3
- `mpeos_dbgLogUDP` DBG-13
- `mpeos_dbgStatusGetTypes` DBG-14
- `mpeos_dbgStatusRegister` DBG-15
- `mpeos_dbgStatusRegisterInterest` DBG-16
- `mpeos_dbgStatusUnregister` DBG-17
- `mpeos_dbgStatusUnregisterInterest` DBG-18
- `mpeos_disp.c` 5-5
- `mpeos_dll.c` 5-3
- `mpeos_dvr.c` 5-4
- `mpeos_event.c` 5-3
- `mpeos_file.c` 5-5
- `mpeos_filesys_ftable_t` FIL-25, PER-3
- `mpeos_filter.c` 5-5
- `mpeos_frontpanel.c` 5-6
- `mpeos_g_logControlTbl` DBG-10
- `mpeos_GfxContext` GFX-24
- `mpeos_GfxFont` GFX-25
- `mpeos_GfxFontFactory` GFX-27
- `mpeos_GfxScreen` GFX-28
- `mpeos_GfxSurface` GFX-30
- `MPEOS_LOG` DBG-4
- `mpeos_main.c` 5-6
- `mpeos_media.c` 5-6

`mpeos_MediaVolume` DVR-13
`mpeos_mem.c` 5-4
`mpeos_memAllocP` MEM-4
`mpeos_memFreeP` MEM-5
`mpeos_memReallocP` MEM-5
`mpeos_pod.c` 5-6
`mpeos_si.c` 5-6
`mpeos_snd.c` 5-6
`mpeos_socket.c` 5-5
`mpeos_Storage` STG-8
`mpeos_storage.c` 5-5
`mpeos_sync.c` 5-4
`mpeos_thread.c` 5-4
`mpeos_time.c` 5-4
`mpeos_uievent.c` 5-5
`mpeos_util.c` 5-4

N

Network

- Test code 6-7

Networking

- Data types and structures NET-23
- Error codes NET-17
- General data types NET-23
- IPv4 data types NET-25
- IPv4-level socket definition options NET-12
- IPv4-specific definitions NET-5
- IPv6 data types NET-26
- IPv6-level socket definition options NET-14
- IPv6-specific definitions NET-7
- Overview NET-1
- Protocol-independent definitions NET-2
- Socket-level socket definition options NET-8
- Supported functions NET-27
- TCP-level socket definition options NET-16

Networking functions

- `mpeos_socketAccept` NET-29
- `mpeos_socketAtoN` NET-31
- `mpeos_socketBind` NET-32
- `mpeos_socketClose` NET-34
- `mpeos_socketConnect` NET-35
- `mpeos_socketCreate` NET-37
- `mpeos_socketFDClear` NET-39
- `mpeos_socketFDIsSet` NET-40
- `mpeos_socketFDSet` NET-41
- `mpeos_socketFDZero` NET-42
- `mpeos_socketGetHostByAddr` NET

- 43
- mpeos_socketGetHostName NE T-45
- mpeos_socketGetHostName NET-46
- mpeos_socketGetLastError NET-47
- mpeos_socketGetOpt NET-48
- mpeos_socketGetPeerName NET-51
- mpeos_socketGetSockName NET-53
- mpeos_socketHtoNL NET-55
- mpeos_socketHtoNS NET-56
- mpeos_socketInit NET-57
- mpeos_socketioctl NET-58
- mpeos_socketListen NET-59
- mpeos_socketNtoA NET-61
- mpeos_socketNtoHL NET-62
- mpeos_socketNtoHS NET-63
- mpeos_socketNtoP NET-64
- mpeos_socketPtoN NET-65
- mpeos_socketRecv NET-67
- mpeos_socketRecvFrom NET-69
- mpeos_socketSelect NET-71
- mpeos_socketSend NET-74
- mpeos_socketSendTo NET-76
- mpeos_socketSetOpt NET-78
- mpeos_socketShutdown NET-81
- mpeos_socketTerm NET-82

- O**
- OCAP
 - Class libraries 1-3
 - Creating third-party OCAP extensions 4-24
 - Extensions 4-18
 - Jni3pExample OCAP extension 4-19
 - Legacy DVR recording extension 4-20
 - Managers 1-3
 - OCAP_KEY_PRESS GFX-9
 - OCAP_KEY_RELEASE GFX-9
 - omake 4-9
 - OpenCable
 - Host Device 2.0 Core Functional Requirements 1-4
 - Operating system
 - Configuration values 5-8
 - DLL values 5-9
 - DVR values 5-9
 - Error code values 5-10
- Event values 5-12
- File system values 5-13
- Graphic values 5-14
- Media values 5-15
- Networking values 5-19
- Source files 5-3
- Storage values 5-36
- Synchronization values 5-37
- Thread values 5-37
- Time values 5-39, 5-41
- Utility values 5-42
- os_Dlmod MOD-3
- os_DlmodData MOD-3
- os_GfxContext GFX-24
- os_GfxFont GFX-26
- os_GfxScreen GFX-29
- os_GfxSurface GFX-33
- os_Mutex GFX-26
- os_SymbolTbl MOD-3
- other directory 4-5
- Overview
 - ait-xxx.properties file 3-46
 - Build environment 4-1
 - Build files 4-11
 - Closed captioning CAP-1
 - Common download CDL-1
 - Contents of packages 1-6
 - Debug DBG-1
 - Display DSP-1
 - Documentation ii-xxxviii
 - DVR DVR-1
 - Event EVT-1
 - File persistent PER-1
 - File System FIL-1
 - Front panel FPL-1
 - Graphics manager GFX-1
 - Hardware ii-xxxvii
 - hostapp.properties file 3-65
 - Host-device applications 5-43
 - Installing software 2-1
 - Media MED-1
 - Memory MEM-1
 - Module support MOD-1
 - MPEOS 5-1
 - MPEOS testing 6-1
 - Networking NET-1
 - POD POD-1
 - Porting kit 1-1
 - Porting layer 1-2
 - Runtime environment 3-1
 - Section filtering FLT-1

| | | | |
|---------------------------|-----------|------------------------------------|------------|
| Software | ii-xxxvii | Overview | 1-2 |
| Sound | SND-I | Strategy | 5-2 |
| Storage manager | STG-I | | |
| Synchronization | SYN-I | | |
| Thread | THR-I | R | |
| Time | TME-I | Requirements | |
| Understanding the | | Hardware | 1-4 |
| implementation | 5-3 | Software | 1-5 |
| Utilities | UTL-I | Runtime environment | 3-1 |
| VBI | VBI-I | | |
| xait.properties file | 3-55 | S | |
| | | Screen layout | |
| | | Background | DSP-2 |
| P | | Graphics | DSP-2 |
| POD | | Overview | DSP-2 |
| Data types and structures | POD-3 | Planes and television display | DSP-3 |
| Error codes | POD-2 | Video | DSP-2 |
| Events | POD-7 | | |
| Overview | POD-1 | SDK | |
| Source files | 5-6 | Configuration | 2-11 |
| Supported functions | POD-8 | Installation | 2-8 |
| POD functions | | Section filtering | |
| mpeos_podGetAppInfo | POD-10 | Data types and structures | FLT-6 |
| mpeos_podGetCCIBits | POD-11 | Definitions | FLT-3 |
| mpeos_podGetFeatureParam | POD-13 | Error codes | FLT-4 |
| mpeos_podGetFeatures | POD-12 | Event Overview | FLT-11 |
| mpeos_podInit | POD-9 | Overview | FLT-1 |
| mpeos_podMMIClose | POD-15 | Supported functions | FLT-15 |
| mpeos_podMMIConnect | POD-16 | | |
| mpeos_podReceiveAPDU | POD-17 | Section filtering functions | |
| mpeos_podRegister | POD-18 | mpeos_filterCancelFilter | FLT-16 |
| mpeos_podSASClose | POD-19 | mpeos_filterCloseDSGTunnel | FLT-17 |
| mpeos_podSASConnect | POD-20 | mpeos_filterGetSectionHandle | FLT-18 |
| mpeos_podSendAPDU | POD-21 | mpeos_filterGetSize | FLT-20 |
| mpeos_podSetFeatureParam | POD-22 | mpeos_filterInit | FLT-21 |
| mpeos_podUnregister | POD-24 | mpeos_filterOpenDSGTunnel | FLT-22 |
| Porting kit | 1-6 | mpeos_filterRegisterAvailability | FLT-23 |
| Installation | 2-2 | mpeos_filterRelease | FLT-25 |
| Overview | 1-1 | mpeos_filterSectionRead | FLT-26 |
| Porting layer | | mpeos_filterSectionRelease | FLT-28 |
| Architecture | 1-2 | mpeos_filterSetFilter | FLT-29 |
| Host device | 1-3 | mpeos_filterShutdown | FLT-32 |
| Java classes | 1-2 | mpeos_filterUnregisterAvailability | F LT-33 |
| JNI | 1-3 | | |
| JNI porting layer | 1-3 | Serial cable | 1-4 |
| JVM | 1-2 | Signaling files | 3-45 |
| MPE APIs | 1-3 | Software | |
| MPE service managers | 1-3 | Installation overview | 2-1 |
| MPEOS porting layer | 1-3 | Overview | ii-xxxvii |
| OCAP class libraries | 1-3 | Requirements | 1-5 |
| OCAP managers | 1-3 | Software requirements | |

Development system I-5
 Host device I-5
 Software tools 4-8
 ActivePerl 4-9
 Ant 4-8
 Build construction tools 4-9
 Build control tools 4-8
 Cygwin 4-9
 omake 4-9
 Target-specific SDKs 4-10
 Sound
 Data types and structures SND-3
 Error codes SND-2
 Overview SND-1
 Supported functions SND-5
 Sound functions
 mpeos_sndCreateSound SND-6
 mpeos_sndDeleteSound SND-7
 mpeos_sndGetDeviceCount SND-8
 mpeos_sndGetDevices SND-9
 mpeos_sndGetDevicesForSound SN D-10
 mpeos_sndGetMaxPlaybacks SND-11
 mpeos_sndGetTime SND-12
 mpeos_sndInit SND-13
 mpeos_sndPlay SND-14
 mpeos_sndSetTime SND-15
 mpeos_sndStop SND-16
 Source files
 Basic operating-system 5-3
 DVR 5-4
 File and communication services 5-5
 Graphic 5-5
 Media 5-5
 Miscellaneous 5-6
 Overview
 Source files 5-3
 POD 5-6
 Storage manager
 Data types and structures STG-3
 Definitions STG-2
 Overview STG-1
 Supported functions STG-10
 Storage manager functions
 mpeos_storageEject STG-11
 mpeos_storageGetDeviceCount ST G-12
 mpeos_storageGetDeviceList STG-13
 mpeos_storageGetInfo STG-14
 mpeos_storagelInit STG-16
 mpeos_storagelInitializeDevice STG-17
 mpeos_storagelDetachable STG-19
 mpeos_storagelDvrCapable STG-20
 mpeos_storagelRemovable STG-21
 mpeos_storageMakeReadyForUse S TG-22
 mpeos_storageMakeReadyToDetach STG-24
 mpeos_storageRegisterQueue STG-25
 Strategy
 Porting 5-2
 Surface DSP-4
 symbolDesc MOD-4
 Synchronization
 Data types and structures SYN-3
 Error codes SYN-2
 Overview SYN-1
 Supported functions SYN-4
 Test code 6-7
 Synchronization functions
 mpeos_condDelete SYN-5
 mpeos_condGet SYN-6
 mpeos_condNew SYN-7
 mpeos_condSet SYN-8
 mpeos_condUnset SYN-9
 mpeos_condWaitFor SYN-10
 mpeos_mutexAcquire SYN-11
 mpeos_mutexAcquireTry SYN-12
 mpeos_mutexDelete SYN-13
 mpeos_mutexNew SYN-14
 mpeos_mutexRelease SYN-15

T

Target device requirements for testing 6-14
 target directory 4-6
 Target-specific SDKs 4-10
 Television I-4
 Test code
 Event 6-5
 Front panel 6-5
 Individual tests 6-5
 Memory 6-6
 Modifying 6-4
 Network 6-7
 Synchronization 6-7
 test_sysRunAllTests() 6-4
 test_sysRunDbgTests() 6-4
 test_sysRunEventTests() 6-4

t
 test_sysRunMathTests() 6-4
 test_sysRunMemTests() 6-4
 test_sysRunStressTests() 6-4
 test_sysRunSyncTests() 6-4
 test_sysRunThreadTests() 6-4
 test_sysRunTimeTests() 6-4
 test_sysRunUtilTests() 6-4
 Time 6-9
 Understanding the testing framework 6-12
 test_sysRunAllTests() 6-4
 test_sysRunDbgTests() 6-4
 test_sysRunEventTests() 6-4
 test_sysRunMathTests() 6-4
 test_sysRunMemTests() 6-4
 test_sysRunStressTests() 6-4
 test_sysRunSyncTests() 6-4
 test_sysRunThreadTests() 6-4
 test_sysRunTimeTests() 6-4
 test_sysRunUtilTests() 6-4
 Testing
 MPEOS 6-1
 TestRunner.c 6-3
 Third-party OCAP extensions 4-24
 Third-party software 5-46
 ARES 5-46
 DirectFB 5-46
 FreeType2 5-46
 Thread
 Definitions THR-3
 Error codes THR-5
 Overview THR-1
 Supported functions THR-7
 Thread functions
 mpeos_threadAttach THR-8
 mpeos_threadCreate THR-9
 mpeos_threadDestroy THR-11
 mpeos_threadGetCurrent THR-12
 mpeos_threadGetData THR-13
 mpeos_threadGetPrivateData THR-14
 mpeos_threadGetStatus THR-15
 mpeos_threadSetData THR-16
 mpeos_threadSetPriority THR-17
 mpeos_threadsetStatus THR-18
 mpeos_threadSleep THR-19
 mpeos_threadYield THR-20
 Time
 Data types and structures TME-3
 Error codes TME-2
 Overview TME-1
 Supported functions TME-4
 Test code 6-9
 Time functions
 mpeos_timeClock TME-5
 mpeos_timeClockTicks TME-6
 mpeos_timeClockToMillis TME-7
 mpeos_timeClockToTime TME-8
 mpeos_timeGet TME-9
 mpeos_timeGetMillis TME-10
 mpeos_timeMillisToClock TME-11
 mpeos_timeSystemClock TME-12
 mpeos_timeTimeToClock TME-13
 mpeos_timeTmToTime TME-14
 mpeos_timeToDate TME-15
 tools directory 4-6

U
 Understanding the implementation 5-3
 Understanding the testing framework 6-12
Utilities
 Data types and structures UTL-4
 Error codes UTL-3
 Events UTL-5
 mpeos_envGet UTL-10
 Overview UTL-1
 Supported functions UTL-6
Utility functions
 mpeos_envInit UTL-11
 mpeos_envSet UTL-12
 mpeos_longJmp UTL-13
 mpeos_registerForPowerKey UTL-14
 mpeos_setJmp UTL-15
 mpeos_stbBoot UTL-16
 mpeos_stbBootStatus UTL-17
 mpeos_stbGetAcOutletState UTL-19
 mpeos_stbGetPowerStatus UTL-20
 mpeos_stbGetRootCerts UTL-21
 mpeos_stblsHdCapable UTL-22
 mpeos_stbSetAcOutletState UTL-23

V
VBI
 Data types and structures VBI-5
 Definitions VBI-2
 Error codes VBI-3
 Events VBI-11
 Overview VBI-1
 Supported functions VBI-14

VBI functions

mpeos_vbiFilterGetParam VBI-15
 mpeos_vbiFilterReadData VBI-16
 mpeos_vbiFilterRelease VBI-18
 mpeos_vbiFilterSetParam VBI-19
 mpeos_vbiFilterStart VBI-20
 mpeos_vbiFilterStop VBI-23
 mpeos_vbiGetParam VBI-24
 mpeos_vbilInit VBI-25
 mpeos_vbiRegisterAvailability VBI-2
 6
 mpeos_vbiShutdown VBI-28
 mpeos_vbiUnregisterAvailability VBI
 -29

X

xait.properties
 Applications 3-57
 Optional fields 3-60
 Required fields 3-57
 Example file 3-64
 Overview 3-55
 Services 3-62
 Optional fields 3-63
 Required fields 3-62
 Transport protocols 3-55
 Version 3-55

