GENIVI Document MG0001
Component Design for IVI LayerManagement 1.0

Version 3.0

15.02.2013

Sponsored by:
GENIVI Alliance

Accepted for release by:
This document has not yet been accepted for release by the GENIVI Alliance Board of Directors.

Abstract:
This document describes the particular components of the Layer Management Service.

Keywords:
GENIVI

GENIVI Alliance, Inc.
2400 Camino Ramon, Suite 375
San Ramon, CA 94583, USA

Participants

This document has been created collaboratively by members of the Media and Graphics expert group. This document is a result of development team graphics activities regarding the specification and standardization of additional needed services inside the graphics stack.

Change History

The following table shows the change history for this specification.

Date	Document Version	Changes
02.07.2010	1.0	Initial version of API, packages and message descriptions added.
07.07.2010	1.1	Added exemplary UML Sequence diagram and chroma key functionality (setChromakeyCommand and message)
02.08.2010	1.2	Added more diagrams showing interaction with other components. Fixed several details of descriptions and api documentation from review
17.12.2010	1.2	Added Genivi Document Number,Figure 7 replaced and Figure 8 added. Some typos corrected
13.04.2011	1.3	Added Requirements tracebility, improved architecture figures and API improvements
06.02.2012	2.0	Updated API References to LayerManagement v0.9.5, updated images, switched to auto-generation of document
15.02.2012	2.0	Updated API References to LayerManagement v1.0 (work in progress)

Contents

1 Introduction

1.1 Purpose

The OEMs within the GENIVI Alliance have been discussing the concept of IVI profiles. The full range of IVI products would be categorised into a number of profiles. A profile would describe the “commodity” functionality within a product segment. Commodity is defined as functionality that is common across the contributing OEMs or functionality that is delivered in a comparable after market consumer device.

The creation of profiles is intended to provide a consistent product target for silicon vendors and GENIVI platform distributors, enabling cost optimisation by the scaling of both the target processor silicon and GENIVI platform distribution.

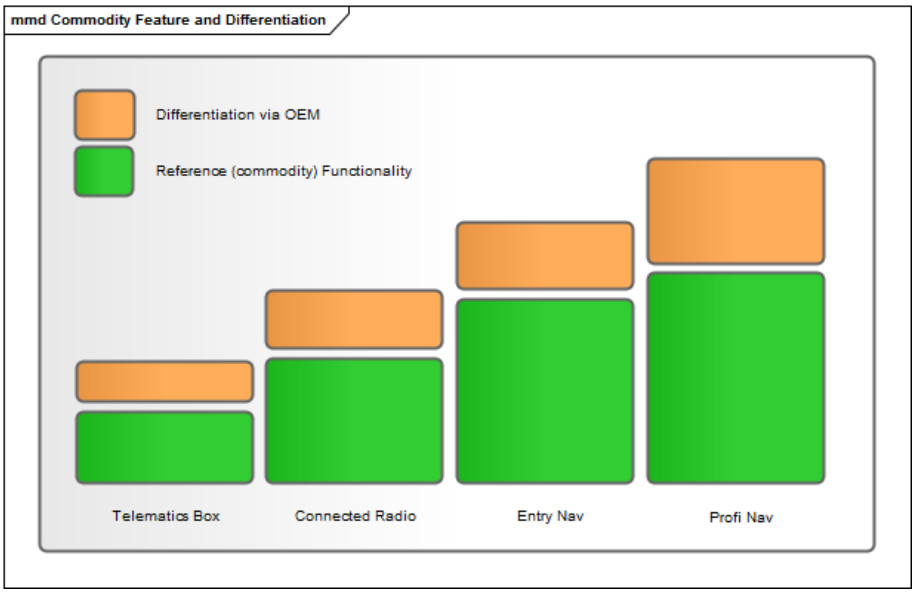

Figure 1: Commodity feature and differentiation

This document identifies the areas of “commodity” features and provides a description of the required functionality. The document will be maintained and further developed within the GENIVI Strategy Council planning work group.

The GENIVI System Architecture Team (SAT) will use the profile descriptions to scope the technical capabilities of the GENIVI platform. The decomposition / further detailing of this document will carried out iteratively between the SC planning group and the SAT.

The commodity functionality within the profiles will describe the functionality to be supported by the GENIVI Reference implementation.

1.2 Purpose of this document

This document contains the detailed design of the Layer Management Service component identified with sufficient detail to allow coding of this component.

2 References

2.1 GENIVI Alliance Documents

- Stakeholder Needs Document (GENIVI Document SND0001)
- Requirements Document (GENIVI Document REQ0001)

2.2 Other Documents

3 Definitions

Name	Definition
Scene	Logical container for multiple screens, layers and surfaces.
Screen	Logical container for multiple layers.
Layer	Logical container for multiple surfaces.
Surface	Graphical content of an application. One application can provide several surfaces.
ILM	IVI Layer Management
IPC	Inter Process Communication.
IVI	In Vehicle Infotainment
RenderOrder	Logical arrangements of surfaces or layers from back to front.

4 Acronyms and Abbreviations

Name	Definition

5 Context

The Layer Management Service is one of a number of components that have been identified in the Graphics Framework Component Model. The components that comprise the Graphics Framework are shown in the following diagram and summarized in the subsequent text.

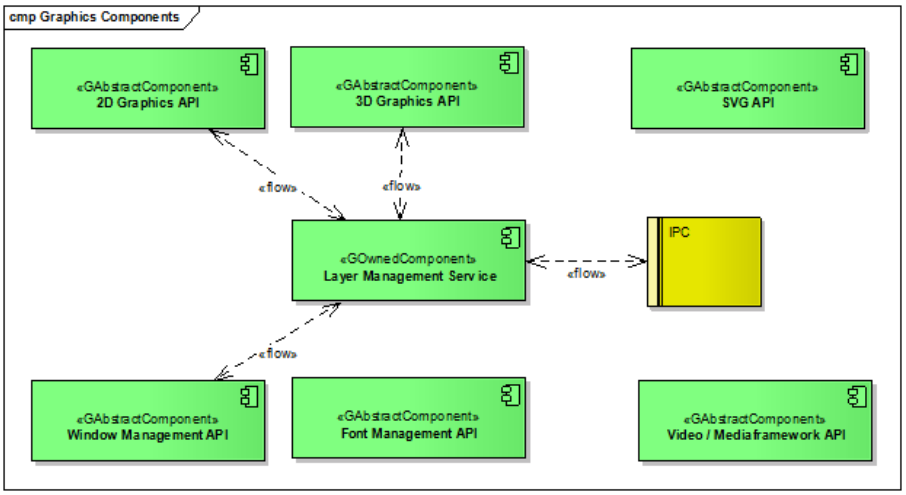

Figure 2: Overall Component Model

The key components are the 2D Graphics API, 3D Graphics API and Window Management API.

5.1 2D Graphics API

The 2D Graphics API takes care of the rendering of 2D graphic primitives, like lines, squares, circles, polygons and splines. Furthermore it will handle skinned applications for OEM branding. On the other side it is possible to reuse that API as a rendering technology of the Layer Management Service.

5.2 3D Graphics API

The 3D Graphics API takes care of the rendering of 3D graphic primitives and supports blending operations for compositing effects. On the other side it is possible to reuse that API as a rendering and compositing technology of the Layer Management Service.

5.3 Window Management API

The window management API is used by applications to display their graphical content. The window management API must provide means to access these graphical application outputs for the Layer Management Service. This requirement is crucial for the Layer Management Service. The specific implementation of this access can have significant impact on the performance of the service. It should be implemented in such a way that no costly copy operations of graphic content are needed. In some cases this additionally poses requirements for graphics hardware or drivers.

6 Specification

6.1 Component Startup

This section describes the mechanisms to start the Layer Management Service component and the actions it takes.

On startup the Layer Management Service shall perform the following tasks:

- Load and instantiate Communication and Rendering packages to be used. These will typically create own threads internally. It must be possible to load these at runtime in order to have maximum flexibility.
- Call start method on loaded packages
- Communication Packages will now typically wait for IPC Calls for creating surfaces, layers etc and arranging themselves. Rendering packages will now typically start rendering all known layers and their surfaces.

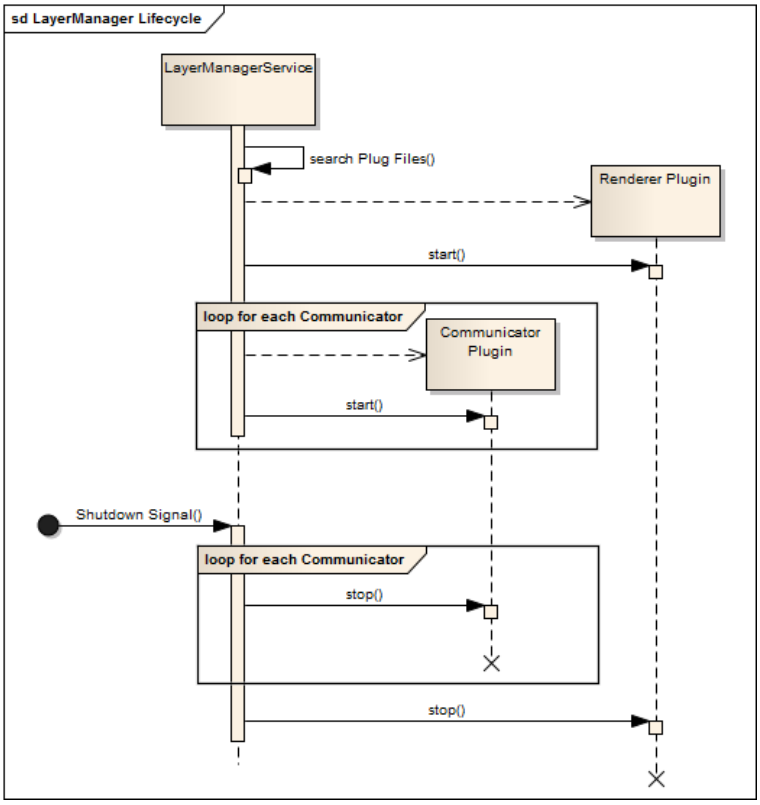

Figure 3: LayerManagerService Lifecycle

6.2 Component Interfaces

6.2.1 Overview

This section describes the Communication interface (Inter Process Communication) between the Layer Management Service component and other components and external systems. This is the interface for other applications to communicate with the Layer Management. This is used to control the Layer Management Service. Design Overview describes internal programming interfaces and is only needed when extending the Layer Management Service with new communication mechanisms or renderers for new platforms. The IPC Message interfaces are described in Layer Management Commands.

6.2.2 Connection Policy

The connection to the service is handled by the used communication packages and as such can not be discussed here generically. The connection policy must be described for each implementation of a communication package to be used.

The list of methods defined in Commands is generic and must always be implemented in communication packages, additional functionality can be provided be individual communication implementations though.

6.2.3 Commands

The IPC Message interfaces are described in Layer Management Commands.

6.3 Requirements

This section describes the non-functional requirements applicable to the Layer Management Service Component. The requirements are split into two groups: those directly met by the component and those where the component is supported by the operational infrastructure.

6.3.1 Non-Functional Requirements

6.3.1.1 Requirement Graphic01: 2D / 3D content simultaneously

The user wants to be able to arrange the view, e.g. in order to have a three dimensional map on the left side and lane guidance information on the right side.

6.3.1.2 Requirement Graphic02: Changing the application layout inside of a HMI system

The user wants to be able to change the layout of the displayed applications of the HMI system. For example he sometimes wants to display In-Vehicle Information or Entertainment Details on the right side of a map displayed by the navigation system or vice versa. Furthermore - in dedicated situations - the content delivered by e.g. the rear view application while reversing has to be on top of other applications without losing menu content information provided by the HMI System.

6.3.1.3 Requirement Graphic03: Display navigation information on a second display

Rear seat passenger wants to see current navigation relevant information, like map showing overview of current route, time and distance to destination and more.

6.3.1.4 Requirement Graphic04: Making screenshot of head unit display

When performing evaluation of the system on bench or road test, tester might need to take a screenshot of the actual screen content.

6.3.1.5 Requirement Graphic05: Showing Additional Information on Top

In some situations the user wants to display additional information on top of e.g. a map provided by the navigation system. This information may include:

- Points of interests
- Lane and distance information during reversing
- Pedestrian and obstacle distance during drive at night

- Speed limit information
- On Screen Display Menu Information

6.3.1.6 Requirement Graphic06: Top of market experience while watching different application-content and additional information.

Today's HMI systems have to be integrate different applications depending on the end-user's need. Typical Applications are which have to be integrated are:

- Navigation application including 2D and 3D Map viewer
- Television and video
- Video content while reversing
- Graphical feedback while reversing
- Browsing the internet
- Entertainment Details
- Vehicle information
- Assistant information
- Interactive Vehicle Manual
- Telephony Application

Therefore the user wants to have a top of market experience while watching and using application content and assistant information, without any disturbance (frame drop, unsmooth displayed animations, response delay on interaction) of the displayed content of the HMI system.

6.3.1.7 Requirement Graphic07: Using different application content

The needs of end-users regarding HMI systems can have a wide variety. Therefore they range from only listening to audio and watching on board vehicle information to watching video, browsing the Internet and using the navigation system simultaneously. Hence today's HMI systems have to integrate different applications depending on the end-user's needs. Typical Applications which have to be integrated are:

- Navigation application including 2D and 3D Map viewer
- Television and video
- Video content while reversing
- Graphical feedback while reversing
- Browsing the internet
- Entertainment Details
- Vehicle information
- Assistant information
- Interactive Vehicle Manual
- Telephony Application

The user wants to use these different applications on a HMI System in parallel.

6.3.1.8 Requirement Graphic08: Showing screen content of connected CE device

User may want to connect smartphone device and see screen content of the device on the head unit display.

Sound output of the connected device should be redirected thru car audio device, input from the head unit should be redirected to the smartphone device.

6.3.2 Requirements placed on the Operational Infrastructure

- Access to graphical content of applications managed by the Window Management API.

7 Functional Overview

The Layer Management Service provides means to arrange the output of graphical applications on the platform in a two dimensional fashion. This Service takes care of which application is shown by the end user needs. Therefore it is possible to change the content of the shown application during runtime, sometimes it is required that different application content is on top of other applications. An important aspect is that a defined interface is given to be able to change the application layout during runtime.

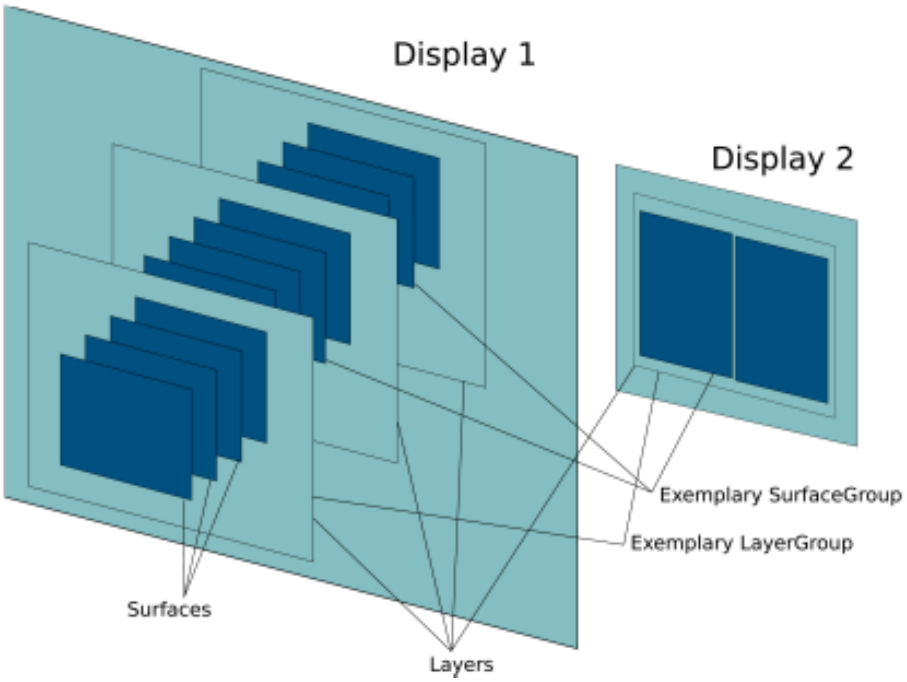

Figure 4: Relation of displays, surfaces and layers (TODO: update picture - remove groups)

The Types Surface and Layer are logical entities contained in corresponding classes within the layermanager. They are not platform dependant, they are just logical entities or data containers keeping the values for opacity, position etc. The class Surface contains a pointer to a platform dependant “PlatformSurface” type with can store platform dependant data for renderers (Window handles for example).

"UML sequence example" diagram shows an exemplary sequence of actions. The LayerManagement control is started, which in turn creates a communicator and starts it. A management application is run and creates an initial layer, for example for third party applications. The management applications then – at a

later point in time – receives an event that a new application has been started or a new window has been created within the window framework for example and creates a Layer Management surface for this application window. Then again later an external event is received which causes the management application to want to reorder the visible output of the Layer Management and sends a SetVisibilityCommand to hide the layer with user applications or to show a notification layer.

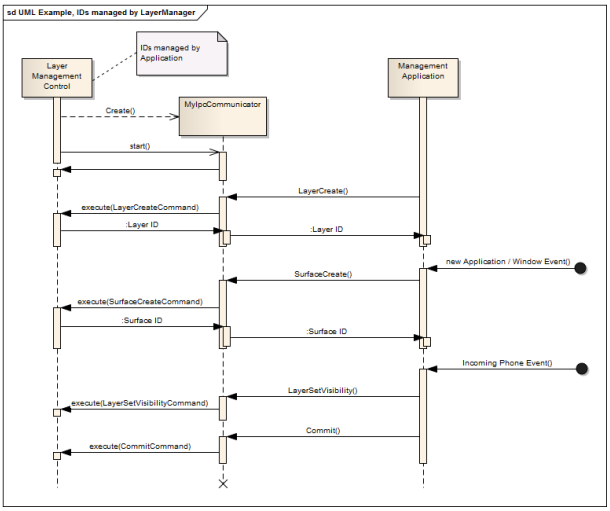

Figure 5: UML Sequence example, IDs managed by LayerManagement

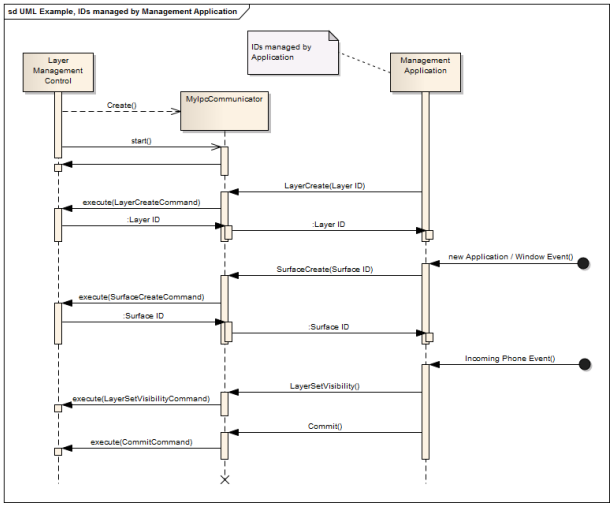

Figure 6: UML Sequence example, IDs managed by Application

Similarly the management application would set multiple properties of multiple surfaces and layers in order to react to user input (change to an application or switch to a “TV mode” etc).

The Layer Management can be used in two scenarios:

7.1 Scenario A: LayerManagement without Central Control Instance

This scenario uses no master to control the LayerManagement setup. All applications talk to the Layer Management themselves and configure their output.

Course of events:

- (1) Application creates window using native window system
- (2) Platform dependant renderer gets new window event from window system
- (3) Gets native window handle
- (4) Application registers/requests logical surface from Layer Management including native window id, width, height and pixelformat of the surface.The renderer uses the given native window id to access the graphical content, this content is associated with the logical surfaceid.
- (5) Layermanager returns the newly created surface identifier
- (6) Application uses this identifier to set properties of its surface

7.2 Scenario B: LayerManagement with Central Control Instance

A central control instances sets up the LayerManagement configuration. It has full control, which applications are shown and the way they are rendered.

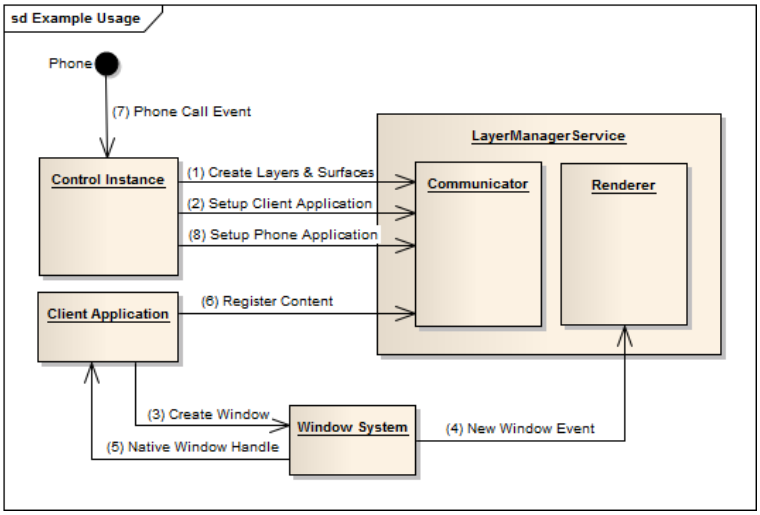

Figure 7: Example with Central Control Instance

Course of events:

- (1) The control instance creates all Layers and Surfaces of the scene according to their properties including native window id, width, height and pixelformat of the surface.
- (2) The control instance configures the scene to show the client application
- (3) Application creates window using native window system

- (4) Platform dependant renderer gets new window event from window system
- (5) Gets native window handle
- (6) The client application registers its native window at the LayerManager. The renderer uses the given native window id to access the graphical content, this content is associated with the logical surfaceid that was created by the control instance.

Now the client application is rendered on screen.

- (7) An external event occurs (e.g. phone call) at the control instance
- (8) Control instance decides to change the graphical arrangement to blend out all applications except telephone application

Now only the phone application is rendered on screen.

8 Design Overview

8.1 Packages

The Layer Management Service is composed of the following packages:

- Layer Management
- One Renderer package
- One or more Communicator packages

The Layer Management Service component makes use of the following external packages provided by the application framework:

- Renderer and Communicator packages which in turn depend on other packages/frameworks for the communication (e.g. middle ware) or the device dependent rendering (e.g. graphic frameworks)

The diagram below shows the relationships between the packages:

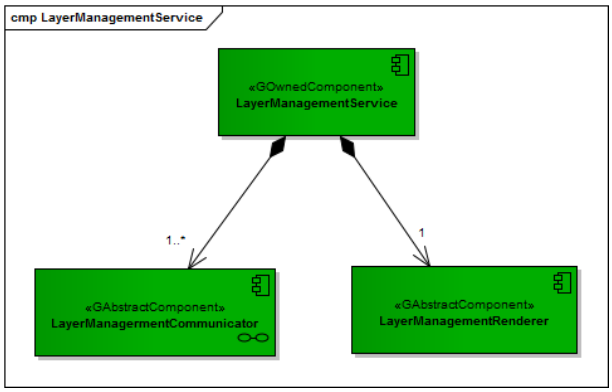

Figure 8: Layer Management Packages

The diagram below shows the interaction between the service and the renderer and communication package-
:

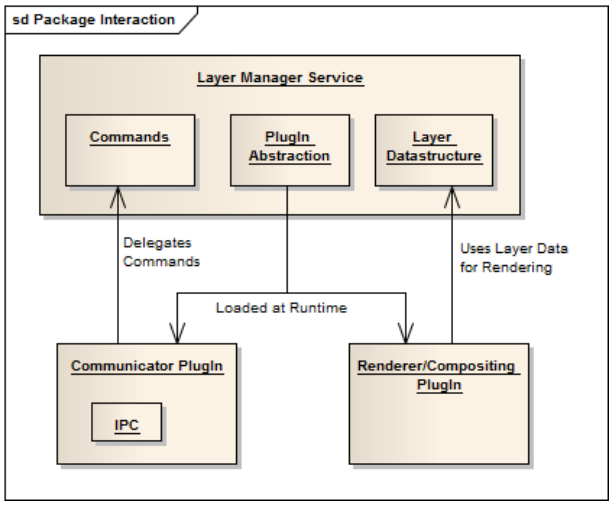

Figure 9: Layer Management Package Interaction

The diagram below shows the command flow from the client application calling the Layer Management Client API - API which each application (including HMI) has to use. to the final screen update:

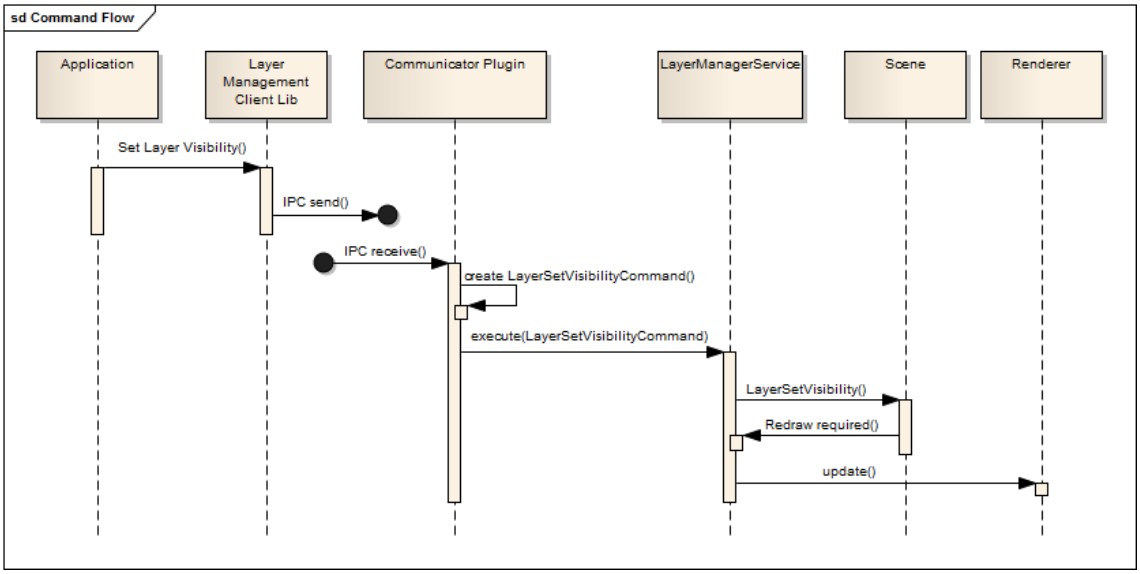

Figure 10: Layer Management Command Flow

The diagram below shows dependencies to all software components used by the LayerManager:

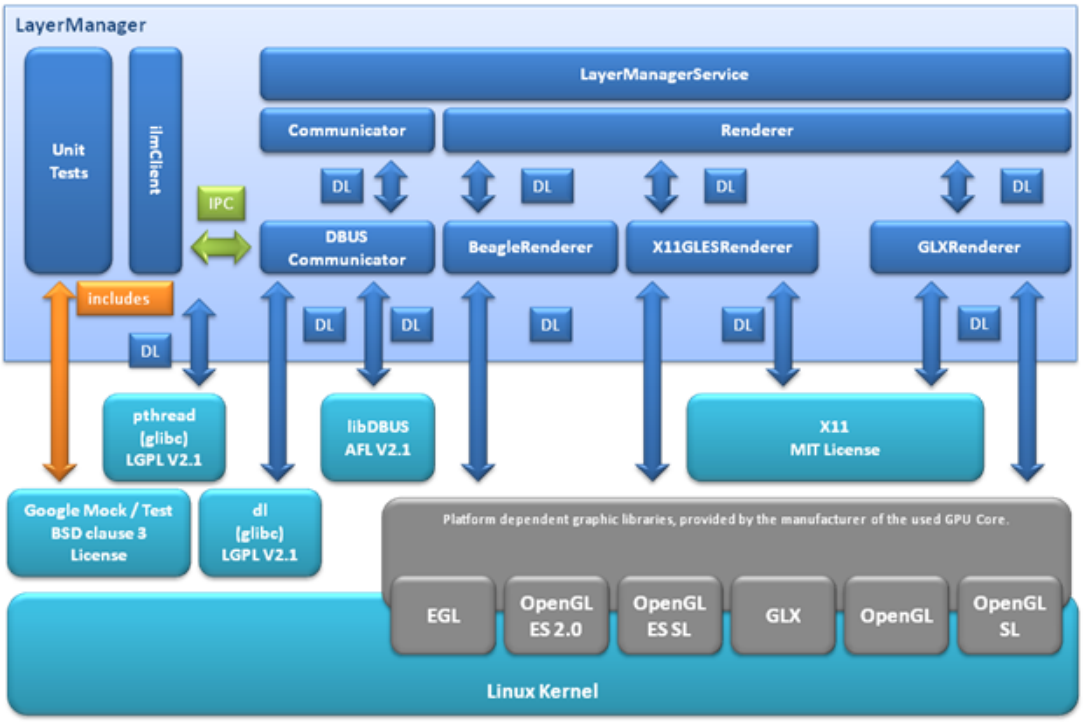

Figure 11: Layer Management Package Dependencies (DL = Dynamic Linking)

9 Service Package

9.1 Overview

This is the main package for the Layer Management Service. Its main purpose is creating all required objects for managing the scene and loading the renderer and communicator plugins.

All configuration options are handled in this component and delegated to the corrensponding instances. This includes the handling of all command line arguments provided during the start of LayerManagement-Service.

Additioanlly, this package provides the definitions for many important APIs, e.g.

- Layer Management Renderer API
- Layer Management Communicator API
- Layer Management Scene API

data types, e.g.

- Layer
- Surface
- Shader

and provides implementations for common classes, e.g.

- Log
- Rectangle
- Vector2

9.2 Object Model

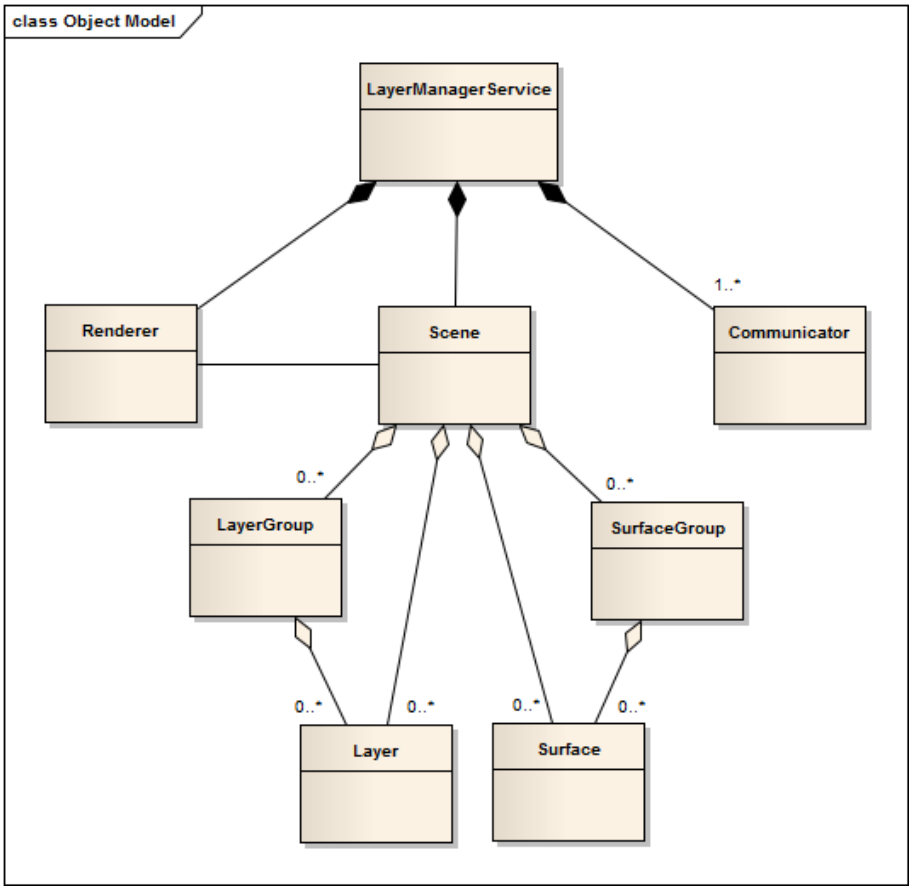

Figure 12: Layer Management Service Control Package

9.3 Layer Management Service

9.3.1 Description

The control is responsible for loading communication and renderer packages to be used. The control initiates the main class, which in turn contains and manages the scene with the list of layers and their surfaces through the Scene object. The renderer packages are given access to these lists by the control and the communication packages must be able to obtain information about properties requested by clients (e.g. “SurfaceGetVisibility”).

9.3.2 Public Interface

The interface of the Service package is described in more detail in Layer Management Service API.

10 Scene Package

10.1 Overview

The scene is an entity for managing the list of screens, layers, their surfaces and the respective properties. It is passed to the render packages so it can be used to iterate through the screens, layers and surfaces and render these in the required render order.

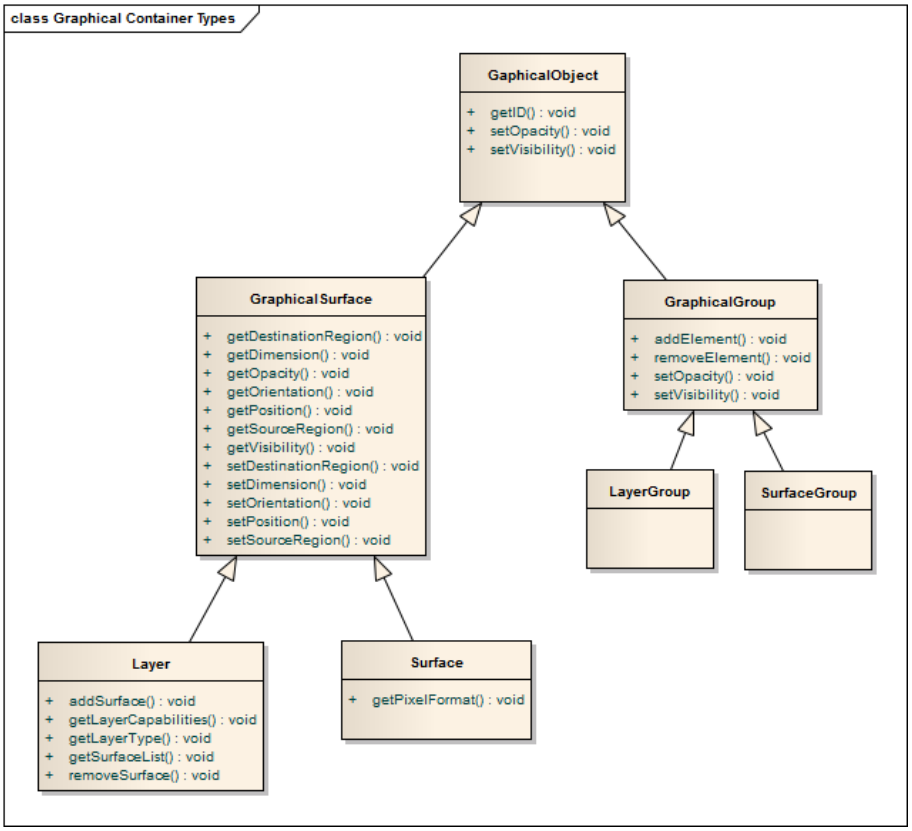

Figure 13: Class Diagram of Internal Container Types

10.2 Public Interface

The interface of the Scene package is described in more detail in Layer Management Scene API.

11 Communications Package

11.1 Overview

There is no direct communication to the Layer Management service. All communication must be performed through communication libraries loaded. These libraries can implement communication through middleware etc. The communication libraries encapsulate these dependencies and relay command objects similar to a command design pattern to the control component of the Layer Management Service. This way the service has no dependencies towards certain ways of communication and the usage of specific communication libraries can be decided at runtime.

On the client side it is recommended to use the Layer Management Client API - API which each application (including HMI) has to use. for communication with the LayerManagerService, or to be more precise, with the loaded Communicator Plugin of the LayerManagerService. The LayerManagment client library implements an abstraction layer hiding the technical details of the underlying communication technology. This enables client applications to be used with different communication technologies implemented in LayerManager Communicator plugins.

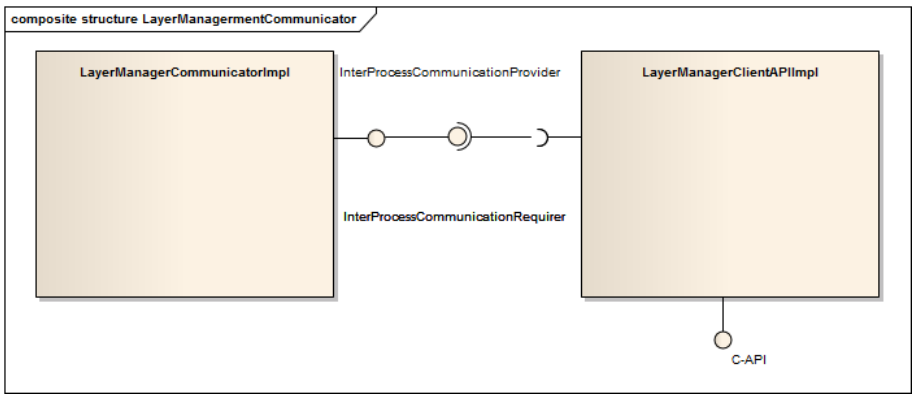

Figure 14: Layer Manager Communicator Structure

The general procedure for communicators is to establish their specific communication (IPC, proprietary method, specific bus etc) and provide the message interface described in chapter 6. When receiving commands on this communication channel the package builds one or more of the command objects described below and calls the execute method of the layermanager with these command objects. The command objects contain everything which is needed to execute the command, the parameters, the type of command etc. Changes to the scene elements and their properties are then executed within the layermanager.

An implementation of a communication library must subclass ICommunicator and implement the inherited start() and stop() methods, as well as a way to load the library dynamically at runtime.

The layermanager searches the provided communicator shared library for two entry points, which are both mandatory for a communicator library. Their name is specified by the following naming scheme:

- `ICommunicator* create<Library_Name>(ICommandExecutor *)`
- `void destroy<Library_Name>(<Library_Name>*)`

In order to be loadable by the layermanager, the created shared library must provide both of these functions.

This component is to be provided by the platform supplier.

The GENIVI Consortium does not mandate any specific implementation. The only constraint is that the proposed solution needs to fulfill the API specification. Furthermore each Communicator Implementation

has to provide a C-API for the Client applications (like HMI, Browser, Navigation) to hide the used Inter-ProcessCommunication scheme.

11.2 Example: Create the communication library “MyCommunicator”

(1) Create the class MyCommunicator, which inherits ICommunicator

(2) Implement the virtual class functions

- `virtual bool start(void)`
- `virtual void stop(void)`

(3) Create the static functions (see example source code below)

- ```
  extern "C"
  ICommunicator* createMyCommunicator(ICommandExecutor
        * pExecutor)
  {
      return new MyCommunicator(pExecutor);
  }
  ```

- ```
  extern "C"
  void destroyMyCommunicator(MyCommunicator* pCommunicator)
  {
      delete pCommunicator;
  }
  ```

(4) Implement communication layer of “MyCommunicator”

(5) Link the implementation to a shared library called “libMyCommunicator.so”

11.3 Reference Implementation

The LayerManagement package contains a reference implementation for a communicator, which internally can use DBUS or TCP/IP.

The source code is available in the

```
<package_root>/LayerManagerPlugins/Communicators/GenericCommunicator
```

directory.

11.4 Public Interface

The interface of the Communicator package is described in more detail in Layer Management Communicator API.

11.5 Client Interface

The interface of the Layer Manager Client package is described in more detail in Layer Management Client API - API which each application (including HMI) has to use..

11.6 Command Object Reference

The description of all available Layer Manager Commands is available in Layer Management Commands.

12 Renderer Package

12.1 Overview

The Layer Management Service does not provide rendering functionality on its own. The rendering of layers and their respective surfaces is always handled by rendering libraries. These are typically device dependant because of possible dependencies on graphics hardware or displays for example.

A typical implementation of a renderer will use the pointer to the scene given in the constructor to access the list of current layers and their respective surfaces. In its own thread it will use the information in the scene to render its content. In most cases the renderer will need specific platform information for each surface in order to access the actual graphical content of the platform (e.g. native window handles or memory addresses). For this reason a renderer can append an instance of a subclass of the type PlatformSurface to a Surface object. This can be used to save information like window handles into these objects as these subclasses can be defined by the renderer implementation.

An implementation of a renderer library must subclass BaseRenderer and implement the inherited start(), stop() and doScreenShot() methods, as well as a way to load the library dynamically at runtime.

The layermanager searches the provided renderer shared library for two entry points, which are both mandatory for a renderer library. Their name is specified by the following naming scheme:

- `BaseRenderer* create<Library_Name>(Scene*)`
- `void destroy<Library_Name>(<Library_Name>*)`

In order to be loadable by the layermanager, the created shared library must provide both of these functions.

12.2 Architecture Overview

The reference renderer plugins are assembled using re-usable modules implementing different aspects like window systems or texture binders.

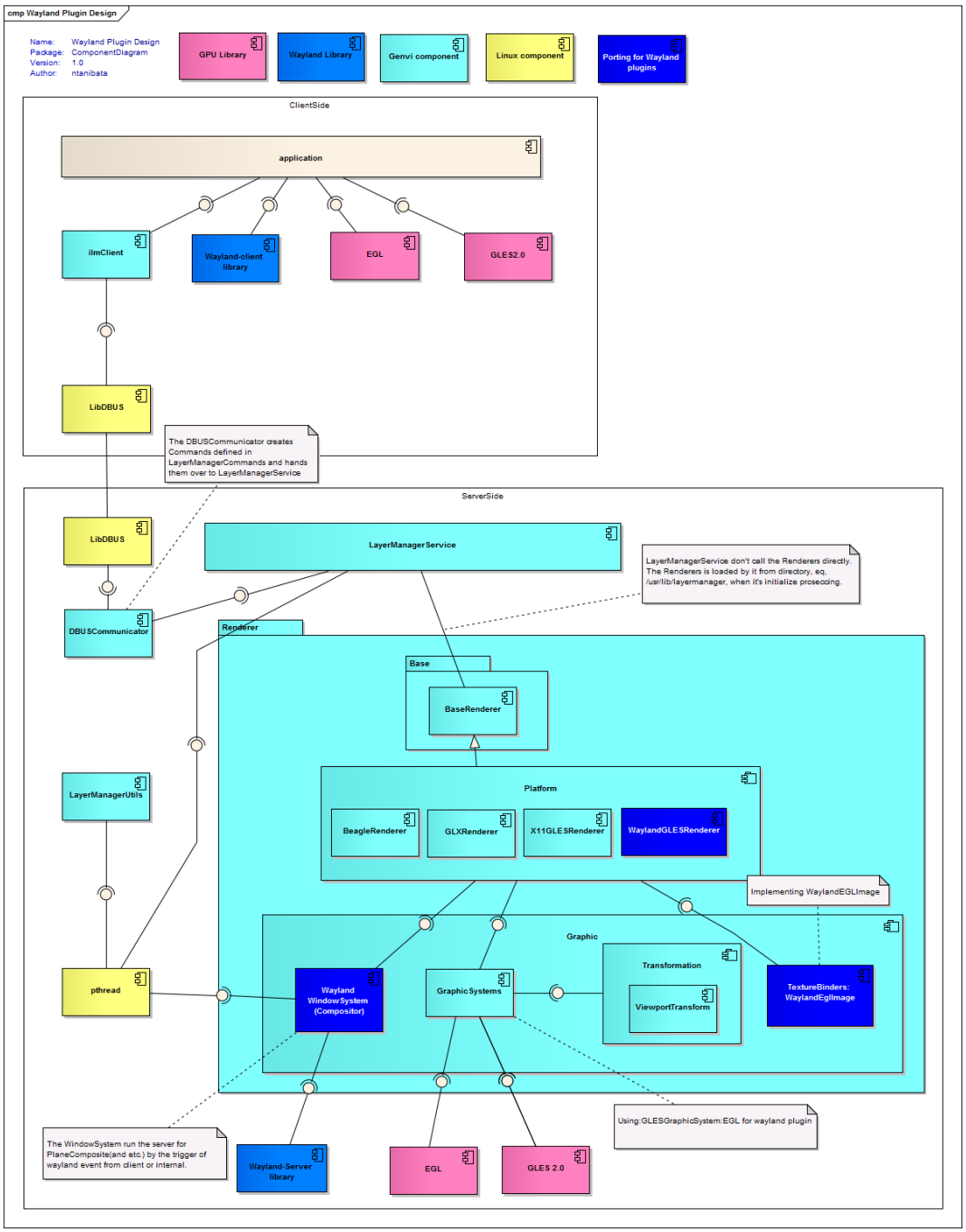

Figure 15: Renderer Architecture Overview

In order to implement a new renderer it often is sufficient to implement only a small amount of modules to switch to a different platform while using ready-to-use modules provided by the reference implementation.

12.3 Example: Create the renderer library “MyRenderer”

(1) Create the class MyRenderer, which inherits BaseRenderer

(2) Implement the virtual class functions

- `virtual bool start(void)`
- `virtual void stop(void)`
- `virtual void doScreenShot(std::string fileToSave)`

(3) Create the static functions (see example source code below)

```
extern "C"
BaseRenderer* createMyRenderer(Scene* pScene){
    return new MyRenderer(pScene);
}
```

```
extern "C"
void destroyMyRenderer(MyRenderer* pRenderer)
{
    delete pRenderer;
}
```

(4) Implement rendering of “MyRenderer”

(5) Link the implementation to a shared library called “libMyRenderer.so”

12.4 Reference Implementation

The LayerManagement package contains two reference implementations for renderers. One is based on OpenGL, the other is based on OpenGL ES 2.0. Both implementations rely on the X11 backend server.

The source code for the OpenGL/X11-based reference renderer is available in the

```
<package_root>/LayerManagerPlugins/Renderers/Platform/GLXRenderer
```

directory.

The source code for the OpenGL ES 2.0/X11-based reference renderer is available in the

```
<package_root>/LayerManagerPlugins/Renderers/Platform/X11GLESRenderer
```

directory.

12.5 Public Interface

The interface of the Renderer package is described in more detail in Layer Management Renderer API.

13 Implementation Notes

The Layer Management service consists of platform independent and platform dependent components. On each platform a separate renderer and if required a separate communicator have to be implemented.

13.1 Reference Communicator

The GENIVI reference communicator implementation depends on either

- DBUS interface or
- TCP/IP sockets.

13.2 OpenGL Reference Renderer

The OpenGL based renderer uses

- X-Composite
- X-Damage

to access the content of different applications and depends on

- the glx extension GLX_EXT_Texture_from_pixmap
- the blending mode of OpenGL

for compositing.

13.3 OpenGL ES 2.0 Reference Renderer

The OpenGL ES 2.0 based renderer uses

- X-Composite
- X-Damage

to access the content of different applications and depends on

- glEGLImageTargetTexture2DOES
- eglCreateImageKHR
- eglDestroyImageKHR
- support for creating an egl image from an X11 pixmap

for compositing.

14 Testing

14.1 Overview

All packages (Communicator, Renderers, Scene and Layermanager) must be tested on their own. The division of the Layer Management Service into these packages provides an easy usage of black box testing.

For a smoke test based approach constellations of communicator packages and renderers can be tested by starting up the Layer Management, inserting automated messages into the communicator channel and comparing the rendered result to the desired output or resulting changes.

14.2 Communicator

All communicator packages should be black box tested by configuring the communicator package under test with a mock Layer Management, then using the provided communicator channel it should be verified, that received messages result in “execute” calls to the Layer Management. This way the wanted functionality of communicator packages can be tested fully.

14.3 Renderers

In a similar way, renderers can be tested by providing certain constellations of layers, surfaces and their properties in a mock Layerlist. This mock layerlist is then given to the renderer under test and then checking for the desired output of the renderer (positioning, overlapping, transparency etc).

14.4 Scene

The implementation of the Scene can also be tested by automatic tests, inserting new layers and surfaces, then changing properties and comparing the results of subsequent calls to getter methods with the desired values.

15 Traceability to the Requirements

16 Constraints and Assumptions

16.1 Solution Constraints

A completely platform independent solution is not possible due to the dependence on a specific platform for rendering and window management. The division of the Layer Management Service addresses this problem by having separate rendering packages for each target platform. The communication packages, main program and scene are not platform dependant. For maximum flexibility the communication and rendering packages must not be known at compile time of the rest of the Layer Management, i.e. they are loaded at runtime and integrated using a defined set of entry points.

16.2 External Factors

When implementing rendering packages special care must be taken with respect to performance, because certain parts of the code will be executed at high frequency per second.

16.3 Assumptions

The Window Management API in use must provide access to the graphical content of individual applications for the rendering package.

17 Future Requirements

18 Howtos and Step-by-Step Guides

18.1 Howto Create a LayerManagement Plugin

18.1.1 Setup Build System

- create folder LayerManagerPlugins/<PluginType>/<PluginName> (referred to as [PLUGIN_DIR])

- add File CMakeLists.txt to [PLUGIN_DIR] using this template:

```
##############################################################################
#
# Copyright <year> <your_company>
#
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#               http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
##############################################################################

cmake_minimum_required (VERSION 2.6)

#=============================================================================
# plugin configuration
#=============================================================================
project(<<PLUGIN_NAME>>)
project_type(PLUGIN)

include_directories(
    include
    ${CMAKE_SOURCE_DIR}/LayerManagerUtils/include
    ${CMAKE_SOURCE_DIR}/LayerManagerBase/include
)

set(LIBS
    LayerManagerUtils
    LayerManagerBase
)

set(SRC_FILES
    <<PLUGIN_SOURCE_FILES>>
)

#=============================================================================
# create plugin
#=============================================================================
add_library(${PROJECT_NAME} ${LIBRARY_BUILDMODE} ${SRC_FILES})

install(TARGETS             ${PROJECT_NAME}
        LIBRARY DESTINATION lib/layermanager/<<PLUGIN_TYPE>>
        ARCHIVE DESTINATION lib/layermanager/static)

#=============================================================================
# external libraries
#=============================================================================
target_link_libraries(${PROJECT_NAME} ${LIBS})

add_dependencies(${PROJECT_NAME} ${LIBS})
```

- fill in your real values for
- <<PLUGIN_NAME>>: any name you like, bu no spaces or special characters except '-' or '_' (e.g. MyNewPlugin_v2_1)
- <<PLUGIN_SOURCE_FILES>>: list of source files to be included in plugin, usually a list like "src/fileA.cpp src/fileB.cpp src/fileC.cpp)
- <<PLUGIN_TYPE>>: may be renderer, communicator, ipcmodule, sceneprovider, healthmonitor

18.1.2 Create a Renderer Plugin

18.1.3 Create a Communicator Plugin

18.1.4 Create a Scene Provider Plugin

18.1.5 Create a Health Monitor Plugin

18.1.6 Create a Ipc Module Plugin

19 LayerManagement Command Frequency Overview

20 LayerManagement Command Action Overview

21 Module Documentation

21.1 Layer Management Service API

This file defines the order of groups in the generated documentation.

Functions

- virtual ICommandExecutor::~ICommandExecutor ()
  default destructor
- virtual bool ICommandExecutor::execute (ICommand *commandToBeExecuted)=0
  Have a command processed.
- virtual bool ICommandExecutor::startManagement ()=0
  start layer management
- virtual bool ICommandExecutor::stopManagement ()=0
  stop layer management
- virtual Scene * ICommandExecutor::getScene (void)=0
  get handle to scene data structure
- virtual RendererList * ICommandExecutor::getRendererList (void)=0
  get list of renderer plugins currently used
- virtual CommunicatorList * ICommandExecutor::getCommunicatorList (void)=0
  get list of communicator plugins currently used
- virtual SceneProviderList * ICommandExecutor::getSceneProviderList (void)=0
  get list of scene provider plugins currently used
- virtual HealthMonitorList * ICommandExecutor::getHealthMonitorList (void)=0
  get list of health monitor plugins currently used
- virtual ApplicationReferenceMap * ICommandExecutor::getApplicationReferenceMap (void)=0
  get map of currently registered applications
- virtual void ICommandExecutor::addApplicationReference (t_ilm_client_handle client, IApplication-Reference *applicationReference)=0
  add application to list of currently registered applications
- virtual void ICommandExecutor::removeApplicationReference (t_ilm_client_handle client)=0
  remove application from list of currently registered applications
- virtual t_ilm_uint ICommandExecutor::getSenderPid (t_ilm_client_handle client)=0

get pid of a connected application

- virtual const char * ICommandExecutor::getSenderName (t_ilm_client_handle client)=0

  get name of a connected application
- virtual const char * ICommandExecutor::getSenderName (unsigned int pid)=0

  get name of a connected application
- virtual unsigned int ICommandExecutor::getLayerTypeCapabilities (const LayerType layertype) const =0

  get capabilities of layer type
- virtual unsigned int ICommandExecutor::getNumberOfHardwareLayers (const unsigned int screenID) const =0

  get number of supported hardware layers for screen
- virtual unsigned int * ICommandExecutor::getScreenResolution (const unsigned int screenID) const =0

  get resolution of screen
- virtual unsigned int * ICommandExecutor::getScreenIDs (unsigned int *length) const =0

  get list of available screens
- virtual void ICommandExecutor::addClientNotification (GraphicalObject *object, t_ilm_notification_mask mask)=0

  add a notification for an updated scene element
- virtual NotificationQueue & ICommandExecutor::getClientNotificationQueue ()=0

  get the list of updated scene elements
- virtual CommandList & ICommandExecutor::getEnqueuedCommands (unsigned int clientPid)=0

  get the list of enqueued commands for a client
- virtual HealthCondition ICommandExecutor::getHealth ()=0

  get system health state

21.1.1 Detailed Description

This file defines the order of groups in the generated documentation. Interface for LayerManagement Command Executors.

Objects who implement this interface can be used to have command objects executed. Communication classes must only know this interface of a class to be able to pass along command objects.

Note

This interface is used to reduce dependency of communicators on the main layermanagement component

21.1.2 Function Documentation

21.1.2.1 virtual void ICommandExecutor::addApplicationReference (t_ilm_client_handle *client*, IApplicationReference * *applicationReference*) [pure virtual]

add application to list of currently registered applications

Parameters

in	*client*	handle to connected client application
in	*applicationReference*	pointer to application object

Implemented in Layermanager.

21.1.2.2 virtual void ICommandExecutor::addClientNotification (GraphicalObject * *object*, t_ilm_notification_mask *mask*) [pure virtual]

add a notification for an updated scene element

Parameters

in	*object*	pointer to updated scene element, e.g. layer or surface
in	*mask*	bitmask indicating which property of scene element was udpated

Implemented in Layermanager.

21.1.2.3 virtual bool ICommandExecutor::execute (ICommand * *commandToBeExecuted*) [pure virtual]

Have a command processed.

Parameters

in	*commandToBe-Executed*	The command to be processed

Returns

TRUE: execution of command successful
FALSE: execution of command failed

Implemented in Layermanager.

21.1.2.4 virtual ApplicationReferenceMap* ICommandExecutor::getApplicationReferenceMap (void) [pure virtual]

get map of currently registered applications

Returns

map of currently registered applications

Implemented in Layermanager.

21.1.2.5 virtual NotificationQueue& ICommandExecutor::getClientNotificationQueue ( ) [pure virtual]

get the list of updated scene elements

Returns

reference to current list of updated scene elements

Implemented in Layermanager.

21.1.2.6 virtual CommunicatorList* ICommandExecutor::getCommunicatorList (void) [pure virtual]

get list of communicator plugins currently used

Returns

Pointer to internal list of communicators

Implemented in Layermanager.

21.1.2.7 virtual CommandList& ICommandExecutor::getEnqueuedCommands (unsigned int *clientPid*) [pure virtual]

get the list of enqueued commands for a client

Parameters

in	*clientPid*	process id of client

Returns

Reference to command list for client

Implemented in Layermanager.

21.1.2.8 virtual HealthCondition ICommandExecutor::getHealth ( ) [pure virtual]

get system health state

Returns

system health condition

Implemented in Layermanager.

21.1.2.9 virtual HealthMonitorList* ICommandExecutor::getHealthMonitorList (void) [pure virtual]

get list of health monitor plugins currently used

Returns

Pointer to internal list of health monitors

Implemented in Layermanager.

21.1.2.10 virtual unsigned int ICommandExecutor::getLayerTypeCapabilities (const LayerType *layertype*) const [pure virtual]

get capabilities of layer type

Parameters

in	*layertype*	layer type

Returns

bitmask with capability flags set according to layer type capabilities

Implemented in Layermanager.

21.1.2.11 virtual unsigned int ICommandExecutor::getNumberOfHardwareLayers (const unsigned int *screenID*) const [pure virtual]

get number of supported hardware layers for screen

Parameters

in	*screenID*	id of screen

Returns

number of supported hardware layers for screen

21.1.2.12 virtual RendererList* ICommandExecutor::getRendererList (void) [pure virtual]

get list of renderer plugins currently used

Returns

Pointer to list of renderer plugins

Implemented in Layermanager.

21.1.2.13 virtual Scene* ICommandExecutor::getScene (void) [pure virtual]

get handle to scene data structure

Returns

Pointer to the internal scene data structure

Implemented in Layermanager.

21.1.2.14 virtual SceneProviderList* ICommandExecutor::getSceneProviderList (void) [pure virtual]

get list of scene provider plugins currently used

Returns

Pointer to internal list of communicators

Implemented in Layermanager.

21.1.2.15 virtual unsigned int* ICommandExecutor::getScreenIDs (unsigned int * *length*) const [pure virtual]

get list of available screens

Parameters

out	*length*	of returned array

Returns

list of screen ids

21.1.2.16 virtual unsigned int* ICommandExecutor::getScreenResolution (const unsigned int *screenID*) const [pure virtual]

get resolution of screen

Parameters

in	*screenID*	id of screen

Returns

array of integer with width and height of screen

21.1.2.17 virtual const char* ICommandExecutor::getSenderName (t_ilm_client_handle *client*) [pure virtual]

get name of a connected application

Parameters

in	*client*	client handle to get process name for

Implemented in Layermanager.

21.1.2.18 virtual const char* ICommandExecutor::getSenderName (unsigned int *pid*) [pure virtual]

get name of a connected application

Parameters

in	*pid*	process id to get process name for

Implemented in Layermanager.

21.1.2.19 virtual t_ilm_uint ICommandExecutor::getSenderPid (t_ilm_client_handle *client*) [pure virtual]

get pid of a connected application

Parameters

in	*client*	client handle to get process id for

Implemented in Layermanager.

21.1.2.20 virtual void ICommandExecutor::removeApplicationReference (t_ilm_client_handle *client*) [pure virtual]

remove application from list of currently registered applications

Parameters

in	*client*	handle to connected client application

Implemented in Layermanager.

21.1.2.21 virtual bool ICommandExecutor::startManagement ( ) [pure virtual]

start layer management

Returns

TRUE: start management successful
FALSE: start management failed

Implemented in Layermanager.

21.1.2.22 virtual bool ICommandExecutor::stopManagement ( ) [pure virtual]

stop layer management

Returns

TRUE: stopped management successfully
FALSE: stopping management failed

Implemented in Layermanager.

21.1.2.23 virtual ICommandExecutor::~ICommandExecutor ( ) [inline], [virtual]

default destructor

21.2 Layer Management Common API - API used by all clients implementing lifecycle and communication

Functions

- ilmErrorTypes ilm_init ()

  Initializes the IVI LayerManagement Client.
- t_ilm_bool ilm_isInitialized ()

  Returns initialization state of the IVI LayerManagement Client.
- ilmErrorTypes ilm_commitChanges ()

  Commit all changes and execute all enqueued commands since last commit.
- ilmErrorTypes ilm_destroy ()

  Destroys the IVI LayerManagement Client.

21.2.1 Detailed Description

21.2.2 Function Documentation

21.2.2.1 ilmErrorTypes ilm_commitChanges ( )

Commit all changes and execute all enqueued commands since last commit.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.2.2.2 ilmErrorTypes ilm_destroy ( )

Destroys the IVI LayerManagement Client.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not be closed or was not initialized.

21.2.2.3 ilmErrorTypes ilm_init ( )

Initializes the IVI LayerManagement Client.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if a connection can not be established to the services.

21.2.2.4 t_ilm_bool ilm_isInitialized ( )

Returns initialization state of the IVI LayerManagement Client.

Returns

true if client library is initialized
false if client library is not initialized

21.3 Layer Management Client API - API which each application (including HMI) has to use.

Classes

- struct ilmSurfaceProperties

  Typedef for representing a the surface properties structure.

Macros

- #define ILM_TRUE 1u

  Represent the logical true value.
- #define ILM_FALSE 0u

  Represent the logical false value.
- #define ILM_ERROR_STRING(x)

  Macro to translate error codes into error description strings.

Typedefs

- typedef enum e_ilmErrorTypes ilmErrorTypes

  Enumeration on possible error codes.
- typedef enum e_ilmPixelFormat ilmPixelFormat

  Enumeration for supported pixelformats.
- typedef unsigned int ilmInputDevice

  Identifier of different input device types. Can be used as a bitmask.
- typedef t_ilm_uint t_ilm_layer

  Typedef for representing a layer.
- typedef t_ilm_uint t_ilm_surface

  Typedef for representing a surface.
- typedef t_ilm_uint t_ilm_display

  Typedef for representing a display number.
- typedef t_ilm_ulong t_ilm_nativehandle

  Typedef for representing a native window handle.
- typedef t_ilm_char ∗ t_ilm_string

  Typedef for representing a ascii string.
- typedef t_ilm_const_char ∗ t_ilm_const_string

  Typedef for representing a const ascii string.

Enumerations

- enum e_ilmErrorTypes {
  ILM_SUCCESS = 0,
  ILM_FAILED = 1,
  ILM_ERROR_INVALID_ARGUMENTS = 2,
  ILM_ERROR_ON_CONNECTION = 3,
  ILM_ERROR_RESOURCE_ALREADY_INUSE = 4,
  ILM_ERROR_RESOURCE_NOT_FOUND = 5,
  ILM_ERROR_NOT_IMPLEMENTED = 6,
  ILM_ERROR_UNEXPECTED_MESSAGE = 7 }

Enumeration on possible error codes.

- enum e_ilmPixelFormat {
  ILM_PIXELFORMAT_R_8 = 0,
  ILM_PIXELFORMAT_RGB_888 = 1,
  ILM_PIXELFORMAT_RGBA_8888 = 2,
  ILM_PIXELFORMAT_RGB_565 = 3,
  ILM_PIXELFORMAT_RGBA_5551 = 4,
  ILM_PIXELFORMAT_RGBA_6661 = 5,
  ILM_PIXELFORMAT_RGBA_4444 = 6,
  ILM_PIXEL_FORMAT_UNKNOWN = 7 }

  Enumeration for supported pixelformats.

Functions

- ilmErrorTypes ilm_getPropertiesOfSurface (t_ilm_uint surfaceID, struct ilmSurfaceProperties *pSurfaceProperties)

  Get the surface properties from the Layermanagement.
- ilmErrorTypes ilm_getScreenResolution (t_ilm_uint screenID, t_ilm_uint *pWidth, t_ilm_uint *pHeight)

  Get the screen resolution of a specific screen from the Layermanagement.
- ilmErrorTypes ilm_layerAddSurface (t_ilm_layer layerId, t_ilm_surface surfaceId)

  Add a surface to a layer which is currently managed by the service.
- ilmErrorTypes ilm_layerRemoveSurface (t_ilm_layer layerId, t_ilm_surface surfaceId)

  Removes a surface from a layer which is currently managed by the service.
- ilmErrorTypes ilm_surfaceAddNotification (t_ilm_surface surface, surfaceNotificationFunc callback)

  register for notification on property changes of surface
- ilmErrorTypes ilm_surfaceCreate (t_ilm_nativehandle nativehandle, t_ilm_int width, t_ilm_int height, ilmPixelFormat pixelFormat, t_ilm_surface *pSurfaceId)

  Create a surface.
- ilmErrorTypes ilm_surfaceGetDimension (t_ilm_surface surfaceId, t_ilm_uint *pDimension)

  Get the horizontal and vertical dimension of the surface.
- ilmErrorTypes ilm_surfaceGetVisibility (t_ilm_surface surfaceId, t_ilm_bool *pVisibility)

  Get the visibility of a surface. If a surface is not visible, the surface will not be rendered.
- ilmErrorTypes ilm_surfaceRemove (const t_ilm_surface surfaceId)

  Remove a surface.
- ilmErrorTypes ilm_surfaceRemoveNativeContent (t_ilm_surface surfaceId)

  Remove the native content of a surface.
- ilmErrorTypes ilm_surfaceRemoveNotification (t_ilm_surface surface)

  remove notification on property changes of surface
- ilmErrorTypes ilm_surfaceSetNativeContent (t_ilm_nativehandle nativehandle, t_ilm_int width, t_ilm_int height, ilmPixelFormat pixelFormat, t_ilm_surface surfaceId)

  Set the native content of an application to be used as surface content.
- ilmErrorTypes ilm_surfaceSetSourceRectangle (t_ilm_surface surfaceId, t_ilm_int x, t_ilm_int y, t_ilm_int width, t_ilm_int height)

  Set the area of a surface which should be used for the rendering.
- ilmErrorTypes ilm_UpdateInputEventAcceptanceOn (t_ilm_surface surfaceId, ilmInputDevice devices, t_ilm_bool acceptance)

Set from which kind of devices the surface can accept input events. By default, a surface accept input events from all kind of devices (keyboards, pointer, ...) By calling this function, you can adjust surface preferences. Note that this function only adjust the acceptance for the specified devices. Non specified are keept untouched.

21.3.1 Detailed Description

21.3.2 Macro Definition Documentation

21.3.2.1 #define ILM_ERROR_STRING(*x*)

Value:

```
( (x) == ILM_SUCCESS                      ? "success"
            \
    : (x) == ILM_FAILED                       ? "failed"
              \
    : (x) == ILM_ERROR_INVALID_ARGUMENTS      ? "
      invalid arguments provided"  \
    : (x) == ILM_ERROR_ON_CONNECTION          ? "
      connection error"            \
    : (x) == ILM_ERROR_RESOURCE_ALREADY_INUSE ?
       "resource is already in use"  \
    : (x) == ILM_ERROR_RESOURCE_NOT_FOUND     ? "
      resource was not found"      \
    : (x) == ILM_ERROR_NOT_IMPLEMENTED        ? "
      feature is not implemented"  \
    : (x) == ILM_ERROR_UNEXPECTED_MESSAGE     ? "
      unexpected message received" \
    : "unknown error code"                                                  )
```

Macro to translate error codes into error description strings.

21.3.2.2 #define ILM_FALSE 0u

Represent the logical false value.

21.3.2.3 #define ILM_TRUE 1u

Represent the logical true value.

21.3.3 Typedef Documentation

21.3.3.1 typedef enum e_ilmErrorTypes ilmErrorTypes

Enumeration on possible error codes.

21.3.3.2 typedef unsigned int ilmInputDevice

Identifier of different input device types. Can be used as a bitmask.

21.3.3.3 typedef enum e_ilmPixelFormat ilmPixelFormat

Enumeration for supported pixelformats.

21.3.3.4 typedef t_ilm_const_char* t_ilm_const_string

Typedef for representing a const ascii string.

21.3.3.5 typedef t_ilm_uint t_ilm_display

Typedef for representing a display number.

21.3.3.6 typedef t_ilm_uint t_ilm_layer

Typedef for representing a layer.

21.3.3.7 typedef t_ilm_ulong t_ilm_nativehandle

Typedef for representing a native window handle.

21.3.3.8 typedef t_ilm_char* t_ilm_string

Typedef for representing a ascii string.

21.3.3.9 typedef t_ilm_uint t_ilm_surface

Typedef for representing a surface.

21.3.4 Enumeration Type Documentation

21.3.4.1 enum e_ilmErrorTypes

Enumeration on possible error codes.

Enumerator:

ILM_SUCCESS ErrorCode if the method call was successful

ILM_FAILED ErrorCode if the method call has failed

ILM_ERROR_INVALID_ARGUMENTS ErrorCode if the method was called with invalid arguments

ILM_ERROR_ON_CONNECTION ErrorCode if connection error has occured

ILM_ERROR_RESOURCE_ALREADY_INUSE ErrorCode if resource is already in use

ILM_ERROR_RESOURCE_NOT_FOUND ErrorCode if resource was not found

ILM_ERROR_NOT_IMPLEMENTED ErrorCode if feature is not implemented

ILM_ERROR_UNEXPECTED_MESSAGE ErrorCode if received message has unexpected type

21.3.4.2 enum e_ilmPixelFormat

Enumeration for supported pixelformats.

Enumerator:

ILM_PIXELFORMAT_R_8 Pixelformat value, to describe a 8 bit luminance surface

ILM_PIXELFORMAT_RGB_888 Pixelformat value, to describe a 24 bit rgb surface

ILM_PIXELFORMAT_RGBA_8888 Pixelformat value, to describe a 24 bit rgb surface with 8 bit alpha

ILM_PIXELFORMAT_RGB_565 Pixelformat value, to describe a 16 bit rgb surface

ILM_PIXELFORMAT_RGBA_5551 Pixelformat value, to describe a 16 bit rgb surface, with binary mask

ILM_PIXELFORMAT_RGBA_6661 Pixelformat value, to describe a 18 bit rgb surface, with binars mask

ILM_PIXELFORMAT_RGBA_4444 Pixelformat value, to describe a 12 bit rgb surface, with 4 bit alpha

ILM_PIXEL_FORMAT_UNKNOWN Pixelformat not known

21.3.5 Function Documentation

21.3.5.1 ilmErrorTypes ilm_getPropertiesOfSurface (t_ilm_uint *surfaceID*, struct ilmSurfaceProperties * *pSurfaceProperties*)

Get the surface properties from the Layermanagement.

Parameters

in	*surfaceID*	surface Indentifier as a Number from 0 .. MaxNumber of Surfaces
out	*pSurface-Properties*	pointer where the surface properties should be stored

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not get the resolution.

21.3.5.2 ilmErrorTypes ilm_getScreenResolution (t_ilm_uint *screenID*, t_ilm_uint * *pWidth*, t_ilm_uint * *pHeight*)

Get the screen resolution of a specific screen from the Layermanagement.

Parameters

in	*screenID*	Screen Indentifier as a Number from 0 .. MaxNumber of Screens
out	*pWidth*	pointer where width of screen should be stored
out	*pHeight*	pointer where height of screen should be stored

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not get the resolution.

21.3.5.3 ilmErrorTypes ilm_layerAddSurface (t_ilm_layer *layerId*, t_ilm_surface *surfaceId*)

Add a surface to a layer which is currently managed by the service.

Parameters

in	*layerId*	Id of layer which should host the surface.
in	*surfaceId*	Id of surface which should be added to the layer.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.4 ilmErrorTypes ilm_layerRemoveSurface (t_ilm_layer *layerId*, t_ilm_surface *surfaceId*)

Removes a surface from a layer which is currently managed by the service.

Parameters

in	*layerId*	Id of the layer which contains the surface.
in	*surfaceId*	Id of the surface which should be removed from the layer.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.5 ilmErrorTypes ilm_surfaceAddNotification (t_ilm_surface *surface*, surfaceNotificationFunc *callback*)

register for notification on property changes of surface

Parameters

in	*surface*	id of surface to register for notification
in	*callback*	pointer to function to be called for notification

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.
ILM_ERROR_INVALID_ARGUMENT if the given surface already has notification callback registered

21.3.5.6 ilmErrorTypes ilm_surfaceCreate (t_ilm_nativehandle *nativehandle*, t_ilm_int *width*, t_ilm_int *height*, ilmPixelFormat *pixelFormat*, t_ilm_surface * *pSurfaceId*)

Create a surface.

Parameters

in	*nativehandle*	The native windowsystem's handle for the surface
in	*width*	The original width of the surface
in	*height*	The original height of the surface
in	*pixelFormat*	The pixelformat to be used for the surface
in	*pSurfaceId*	The value pSurfaceId points to is used as ID for new surface;
out	*pSurfaceId*	The ID of the newly created surface is returned in this parameter

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.7 ilmErrorTypes ilm_surfaceGetDimension (t_ilm_surface *surfaceId,* t_ilm_uint * *pDimension*)

Get the horizontal and vertical dimension of the surface.

Parameters

in	*surfaceId*	Id of surface.
out	*pDimension*	pointer to an array where the dimension should be stored. dimension[0]=width, dimension[1]=height

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.8 ilmErrorTypes ilm_surfaceGetVisibility (t_ilm_surface *surfaceId,* t_ilm_bool * *pVisibility*)

Get the visibility of a surface. If a surface is not visible, the surface will not be rendered.

Parameters

in	*surfaceId*	Id of the surface to get the visibility of.
out	*pVisibility*	pointer where the visibility of a surface should be stored ILM_SUCCESS if the surface is visible, ILM_FALSE if the visibility is disabled.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.9 ilmErrorTypes ilm_surfaceRemove (const t_ilm_surface *surfaceId*)

Remove a surface.

Parameters

in	*surfaceId*	The id of the surface to be removed

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.10 ilmErrorTypes ilm_surfaceRemoveNativeContent (t_ilm_surface *surfaceId*)

Remove the native content of a surface.

Parameters

in	*surfaceId*	The ID of the surface

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.11 ilmErrorTypes ilm_surfaceRemoveNotification (t_ilm_surface *surface*)

remove notification on property changes of surface

Parameters

in	*surface*	id of surface to remove notification

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.
ILM_ERROR_INVALID_ARGUMENT if the given surface has no notification callback registered

21.3.5.12 ilmErrorTypes ilm_surfaceSetNativeContent (t_ilm_nativehandle *nativehandle,* t_ilm_int *width,* t_ilm_int *height,* ilmPixelFormat *pixelFormat,* t_ilm_surface *surfaceId*)

Set the native content of an application to be used as surface content.

Parameters

in	*nativehandle*	The native windowsystem's handle for the surface
in	*width*	The original width of the surface
in	*height*	The original height of the surface
in	*pixelFormat*	The pixelformat to be used for the surface
in	*surfaceId*	The ID of the surface

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.13 ilmErrorTypes ilm_surfaceSetSourceRectangle (t_ilm_surface *surfaceId,* t_ilm_int *x,* t_ilm_int *y,* t_ilm_int *width,* t_ilm_int *height*)

Set the area of a surface which should be used for the rendering.

Parameters

in	*surfaceId*	Id of surface.
in	*x*	horizontal start position of the used area
in	*y*	vertical start position of the used area
in	*width*	width of the area
in	*height*	height of the area

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.3.5.14 ilmErrorTypes ilm_UpdateInputEventAcceptanceOn (t_ilm_surface *surfaceId,* ilmInputDevice *devices,* t_ilm_bool *acceptance*)

Set from which kind of devices the surface can accept input events. By default, a surface accept input events from all kind of devices (keyboards, pointer, ...) By calling this function, you can adjust surface preferences. Note that this function only adjust the acceptance for the specified devices. Non specified are keept untouched.

Typicall use case for this function is when dealing with pointer or touch events. Those are normally dispatched to the first visible surface below the coordinate. If you want a different behavior (i.e. forward events to an other surface below the coordinate, you can set all above surfaces to refuse input events)

Parameters

in	*surfaceId*	Identifier of the surface to set the keyboard focus on.
in	*devices*	Bitmask of ilmInputDevice
in	*acceptance*	Indicate if the surface accept or reject input events from the specified devices

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4 Layer Management Control API - Special Control API which has to be used from the HMI/Tooling to control the composition.

Classes

- struct ilmLayerProperties

  Typedef for representing a the layer properties structure.
- struct ilmScreenProperties

  Typedef for representing a the screen properties structure.

Typedefs

- typedef enum e_ilmLayerType ilmLayerType

  Enumeration for supported layertypes.
- typedef enum e_ilmObjectType ilmObjectType

  Enumeration for supported graphical objects.
- typedef enum e_ilmOptimization ilmOptimization

  Enumeration of renderer optimizations.
- typedef enum e_ilmOptimizationMode ilmOptimizationMode

  Enablement states for individual optimizations.
- typedef enum e_ilmOrientation ilmOrientation

  Enumeration for supported orientations of booth, surface and layer.
- typedef t_ilm_uint t_ilm_layercapabilities

  Typedef for representing layer capabilities.

Enumerations

- enum e_ilmLayerType {
  ILM_LAYERTYPE_UNKNOWN = 0,
  ILM_LAYERTYPE_HARDWARE = 1,
  ILM_LAYERTYPE_SOFTWARE2D = 2,
  ILM_LAYERTYPE_SOFTWARE2_5D = 3 }

  Enumeration for supported layertypes.
- enum e_ilmObjectType {
  ILM_SURFACE = 0,
  ILM_LAYER = 1 }

  Enumeration for supported graphical objects.
- enum e_ilmOptimization {
  ILM_OPT_MULTITEXTURE = 0,
  ILM_OPT_SKIP_CLEAR = 1 }

  Enumeration of renderer optimizations.
- enum e_ilmOptimizationMode {
  ILM_OPT_MODE_FORCE_OFF = 0,
  ILM_OPT_MODE_FORCE_ON = 1,
  ILM_OPT_MODE_HEURISTIC = 2,
  ILM_OPT_MODE_TOGGLE = 3 }

  Enablement states for individual optimizations.

- enum e_ilmOrientation {
  ILM_ZERO = 0,
  ILM_NINETY = 1,
  ILM_ONEHUNDREDEIGHTY = 2,
  ILM_TWOHUNDREDSEVENTY = 3 }

  Enumeration for supported orientations of booth, surface and layer.

Functions

- ilmErrorTypes ilm_getPropertiesOfLayer (t_ilm_uint layerID, struct ilmLayerProperties *pLayerProperties)

  Get the layer properties from the Layermanagement.
- ilmErrorTypes ilm_getPropertiesOfScreen (t_ilm_display screenID, struct ilmScreenProperties *pScreenProperties)

  Get the screen properties from the Layermanagement.
- ilmErrorTypes ilm_getNumberOfHardwareLayers (t_ilm_uint screenID, t_ilm_uint *pNumberOfHardwareLayers)

  Get the number of hardware layers of a screen.
- ilmErrorTypes ilm_getScreenIDs (t_ilm_uint *pNumberOfIDs, t_ilm_uint **ppIDs)

  Get the screen Ids.
- ilmErrorTypes ilm_getLayerIDs (t_ilm_int *pLength, t_ilm_layer **ppArray)

  Get all LayerIds which are currently registered and managed by the services.
- ilmErrorTypes ilm_getLayerIDsOnScreen (t_ilm_uint screenID, t_ilm_int *pLength, t_ilm_layer **ppArray)

  Get all LayerIds of the given screen.
- ilmErrorTypes ilm_getSurfaceIDs (t_ilm_int *pLength, t_ilm_surface **ppArray)

  Get all SurfaceIDs which are currently registered and managed by the services.
- ilmErrorTypes ilm_getSurfaceIDsOnLayer (t_ilm_layer layer, t_ilm_int *pLength, t_ilm_surface **ppArray)

  Get all SurfaceIds which are currently registered to a given layer and are managed by the services.
- ilmErrorTypes ilm_layerCreateWithDimension (t_ilm_layer *pLayerId, t_ilm_uint width, t_ilm_uint height)

  Create a layer which should be managed by the service.
- ilmErrorTypes ilm_layerRemove (t_ilm_layer layerId)

  Removes a layer which is currently managed by the service.
- ilmErrorTypes ilm_layerGetType (t_ilm_layer layerId, ilmLayerType *pLayerType)

  Get the current type of the layer.
- ilmErrorTypes ilm_layerSetVisibility (t_ilm_layer layerId, t_ilm_bool newVisibility)

  Set the visibility of a layer. If a layer is not visible, the layer and its surfaces will not be rendered.
- ilmErrorTypes ilm_layerGetVisibility (t_ilm_layer layerId, t_ilm_bool *pVisibility)

  Get the visibility of a layer. If a layer is not visible, the layer and its surfaces will not be rendered.
- ilmErrorTypes ilm_layerSetOpacity (t_ilm_layer layerId, t_ilm_float opacity)

  Set the opacity of a layer.
- ilmErrorTypes ilm_layerGetOpacity (t_ilm_layer layerId, t_ilm_float *pOpacity)

  Get the opacity of a layer.
- ilmErrorTypes ilm_layerSetSourceRectangle (t_ilm_layer layerId, t_ilm_uint x, t_ilm_uint y, t_ilm_uint width, t_ilm_uint height)

  Set the area of a layer which should be used for the rendering. Only this part will be visible.

- ilmErrorTypes ilm_layerSetDestinationRectangle (t_ilm_layer layerId, t_ilm_int x, t_ilm_int y, t_ilm_int width, t_ilm_int height)

  Set the destination area on the display for a layer. The layer will be scaled and positioned to this rectangle for rendering.
- ilmErrorTypes ilm_layerGetDimension (t_ilm_layer layerId, t_ilm_uint *pDimension)

  Get the horizontal and vertical dimension of the layer.
- ilmErrorTypes ilm_layerSetDimension (t_ilm_layer layerId, t_ilm_uint *pDimension)

  Set the horizontal and vertical dimension of the layer.
- ilmErrorTypes ilm_layerGetPosition (t_ilm_layer layerId, t_ilm_uint *pPosition)

  Get the horizontal and vertical position of the layer.
- ilmErrorTypes ilm_layerSetPosition (t_ilm_layer layerId, t_ilm_uint *pPosition)

  Sets the horizontal and vertical position of the layer.
- ilmErrorTypes ilm_layerSetOrientation (t_ilm_layer layerId, ilmOrientation orientation)

  Sets the orientation of a layer.
- ilmErrorTypes ilm_layerGetOrientation (t_ilm_layer layerId, ilmOrientation *pOrientation)

  Gets the orientation of a layer.
- ilmErrorTypes ilm_layerSetChromaKey (t_ilm_layer layerId, t_ilm_int *pColor)

  Sets the color value which defines the transparency value.
- ilmErrorTypes ilm_layerSetRenderOrder (t_ilm_layer layerId, t_ilm_layer *pSurfaceId, t_ilm_int number)

  Sets render order of surfaces within one layer.
- ilmErrorTypes ilm_layerGetCapabilities (t_ilm_layer layerId, t_ilm_layercapabilities *pCapabilities)

  Get the capabilities of a layer.
- ilmErrorTypes ilm_layerTypeGetCapabilities (ilmLayerType layerType, t_ilm_layercapabilities *pCapabilities)

  Get the possible capabilities of a layertype.
- ilmErrorTypes ilm_surfaceInitialize (t_ilm_surface *pSurfaceId)

  Create the logical surface, which has no native buffer associated.
- ilmErrorTypes ilm_surfaceSetVisibility (t_ilm_surface surfaceId, t_ilm_bool newVisibility)

  Set the visibility of a surface. If a surface is not visible it will not be rendered.
- ilmErrorTypes ilm_surfaceSetOpacity (const t_ilm_surface surfaceId, t_ilm_float opacity)

  Set the opacity of a surface.
- ilmErrorTypes ilm_surfaceGetOpacity (const t_ilm_surface surfaceId, t_ilm_float *pOpacity)

  Get the opacity of a surface.
- ilmErrorTypes ilm_SetKeyboardFocusOn (t_ilm_surface surfaceId)

  Set the keyboard focus on a certain surface To receive keyboard events, 2 conditions must be fulfilled: 1- The surface must accept events from keyboard. See ilm_UpdateInputEventAcceptanceOn 2- The keyboard focus must be set on that surface.
- ilmErrorTypes ilm_GetKeyboardFocusSurfaceId (t_ilm_surface *pSurfaceId)

  Get the indentifier of the surface which hold the keyboard focus.
- ilmErrorTypes ilm_surfaceSetDestinationRectangle (t_ilm_surface surfaceId, t_ilm_int x, t_ilm_int y, t_ilm_int width, t_ilm_int height)

  Set the destination area of a surface within a layer for rendering. The surface will be scaled to this rectangle for rendering.
- ilmErrorTypes ilm_surfaceSetDimension (t_ilm_surface surfaceId, t_ilm_uint *pDimension)

  Set the horizontal and vertical dimension of the surface.
- ilmErrorTypes ilm_surfaceGetPosition (t_ilm_surface surfaceId, t_ilm_uint *pPosition)

Get the horizontal and vertical position of the surface.

- ilmErrorTypes ilm_surfaceSetPosition (t_ilm_surface surfaceId, t_ilm_uint *pPosition)

  Sets the horizontal and vertical position of the surface.
- ilmErrorTypes ilm_surfaceSetOrientation (t_ilm_surface surfaceId, ilmOrientation orientation)

  Sets the orientation of a surface.
- ilmErrorTypes ilm_surfaceGetOrientation (t_ilm_surface surfaceId, ilmOrientation *pOrientation)

  Gets the orientation of a surface.
- ilmErrorTypes ilm_surfaceGetPixelformat (t_ilm_layer surfaceId, ilmPixelFormat *pPixelformat)

  Gets the pixelformat of a surface.
- ilmErrorTypes ilm_surfaceSetChromaKey (t_ilm_surface surfaceId, t_ilm_int *pColor)

  Sets the color value which defines the transparency value of a surface.
- ilmErrorTypes ilm_displaySetRenderOrder (t_ilm_display display, t_ilm_layer *pLayerId, const t_ilm_uint number)

  Sets render order of layers on a display.
- ilmErrorTypes ilm_takeScreenshot (t_ilm_uint screen, t_ilm_const_string filename)

  Take a screenshot from the current displayed layer scene. The screenshot is saved as bmp file with the corresponding filename.
- ilmErrorTypes ilm_takeLayerScreenshot (t_ilm_const_string filename, t_ilm_layer layerid)

  Take a screenshot of a certain layer The screenshot is saved as bmp file with the corresponding filename.
- ilmErrorTypes ilm_takeSurfaceScreenshot (t_ilm_const_string filename, t_ilm_surface surfaceid)

  Take a screenshot of a certain surface The screenshot is saved as bmp file with the corresponding filename.
- ilmErrorTypes ilm_SetOptimizationMode (ilmOptimization id, ilmOptimizationMode mode)

  Enable or disable a rendering optimization.
- ilmErrorTypes ilm_GetOptimizationMode (ilmOptimization id, ilmOptimizationMode *mode)

  Get the current enablement for an optimization.
- ilmErrorTypes ilm_layerAddNotification (t_ilm_layer layer, layerNotificationFunc callback)

  register for notification on property changes of layer
- ilmErrorTypes ilm_layerRemoveNotification (t_ilm_layer layer)

  remove notification on property changes of layer

21.4.1 Detailed Description

21.4.2 Typedef Documentation

21.4.2.1 typedef enum e_ilmLayerType ilmLayerType

Enumeration for supported layertypes.

21.4.2.2 typedef enum e_ilmObjectType ilmObjectType

Enumeration for supported graphical objects.

21.4.2.3 typedef enum e_ilmOptimization ilmOptimization

Enumeration of renderer optimizations.

21.4.2.4 typedef enum e_ilmOptimizationMode ilmOptimizationMode

Enablement states for individual optimizations.

21.4.2.5 typedef enum e_ilmOrientation ilmOrientation

Enumeration for supported orientations of booth, surface and layer.

21.4.2.6 typedef t_ilm_uint t_ilm_layercapabilities

Typedef for representing layer capabilities.

21.4.3 Enumeration Type Documentation

21.4.3.1 enum e_ilmLayerType

Enumeration for supported layertypes.

Enumerator:

ILM_LAYERTYPE_UNKNOWN LayerType not known

ILM_LAYERTYPE_HARDWARE LayerType value, to describe a hardware layer

ILM_LAYERTYPE_SOFTWARE2D LayerType value, to describe a redirected offscreen buffer layer

ILM_LAYERTYPE_SOFTWARE2_5D LayerType value, to describe a redirected offscreen buffer layer, which can be rotated in the 3d space

21.4.3.2 enum e_ilmObjectType

Enumeration for supported graphical objects.

Enumerator:

ILM_SURFACE Surface Object Type

ILM_LAYER Layer Object Type

21.4.3.3 enum e_ilmOptimization

Enumeration of renderer optimizations.

Enumerator:

ILM_OPT_MULTITEXTURE Multi-texture optimization

ILM_OPT_SKIP_CLEAR Skip clearing the screen

21.4.3.4 enum e_ilmOptimizationMode

Enablement states for individual optimizations.

Enumerator:

ILM_OPT_MODE_FORCE_OFF Disable optimization

ILM_OPT_MODE_FORCE_ON Enable optimization

ILM_OPT_MODE_HEURISTIC Let renderer choose enablement

ILM_OPT_MODE_TOGGLE Toggle on/and off rapidly for debugging

21.4.3.5 enum e_ilmOrientation

Enumeration for supported orientations of booth, surface and layer.

Enumerator:

ILM_ZERO Orientation value, to describe 0 degree of rotation regarding the z-axis

ILM_NINETY Orientation value, to describe 90 degree of rotation regarding the z-axis

ILM_ONEHUNDREDEIGHTY Orientation value, to describe 180 degree of rotation regarding the z-axis

ILM_TWOHUNDREDSEVENTY Orientation value, to describe 270 degree of rotation regarding the z-axis

21.4.4 Function Documentation

21.4.4.1 ilmErrorTypes ilm_displaySetRenderOrder (t_ilm_display *display*, t_ilm_layer * *pLayerId*, const t_ilm_uint *number*)

Sets render order of layers on a display.

Parameters

in	*display*	Id of display to set the given order of layers.
in	*pLayerId*	array of layer ids
in	*number*	number of layerids in the given array

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.2 ilmErrorTypes ilm_GetKeyboardFocusSurfaceId (t_ilm_surface * *pSurfaceId*)

Get the indentifier of the surface which hold the keyboard focus.

Parameters

out	*pSurfaceId*	Pointer on the a surface identifier

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.3 ilmErrorTypes ilm_getLayerIDs (t_ilm_int * *pLength*, t_ilm_layer ** *ppArray*)

Get all LayerIds which are currently registered and managed by the services.

Parameters

out	*pLength*	Pointer where length of ids array should be stored
out	*ppArray*	Array where the ids should be stored, the array will be allocated inside

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.4 ilmErrorTypes ilm_getLayerIDsOnScreen (t_ilm_uint *screenID*, t_ilm_int * *pLength*, t_ilm_layer ** *ppArray*)

Get all LayerIds of the given screen.

Parameters

in	*screenID*	The id of the screen to get the layer IDs of
out	*pLength*	Pointer where length of ids array should be stored
out	*ppArray*	Array where the ids should be stored, the array will be allocated inside

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.5 ilmErrorTypes ilm_getNumberOfHardwareLayers (t_ilm_uint *screenID*, t_ilm_uint * *pNumberOfHardwareLayers*)

Get the number of hardware layers of a screen.

Parameters

in	*screenID*	id of the screen, where the number of Hardware Layers should be returned
out	*pNumberOf-Hardware-Layers*	pointer where the number of hardware layers should be stored

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not get the resolution.

21.4.4.6 ilmErrorTypes ilm_GetOptimizationMode (ilmOptimization *id*, ilmOptimizationMode * *mode*)

Get the current enablement for an optimization.

Parameters

in	*id*	which optimization to query
out	*mode*	current optimization mode

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.7 ilmErrorTypes ilm_getPropertiesOfLayer (t_ilm_uint *layerID,* struct ilmLayerProperties * *pLayerProperties*)

Get the layer properties from the Layermanagement.

Parameters

in	*layerID*	layer Indentifier as a Number from 0 .. MaxNumber of Layer
out	*pLayer-Properties*	pointer where the layer properties should be stored

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not get the resolution.

21.4.4.8 ilmErrorTypes ilm_getPropertiesOfScreen (t_ilm_display *screenID,* struct ilmScreenProperties * *pScreenProperties*)

Get the screen properties from the Layermanagement.

Parameters

in	*screenID*	screen Indentifier
out	*pScreen-Properties*	pointer where the screen properties should be stored

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not get the resolution.

21.4.4.9 ilmErrorTypes ilm_getScreenIDs (t_ilm_uint * *pNumberOfIDs,* t_ilm_uint ** *ppIDs*)

Get the screen Ids.

Parameters

out	*pNumberOfIDs*	pointer where the number of Screen Ids should be returned
out	*ppIDs*	pointer to array where the IDs should be stored

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not get the resolution.

21.4.4.10 ilmErrorTypes ilm_getSurfaceIDs (t_ilm_int * *pLength,* t_ilm_surface ** *ppArray*)

Get all SurfaceIDs which are currently registered and managed by the services.

Parameters

out	*pLength*	Pointer where length of ids array should be stored
out	*ppArray*	Array where the ids should be stored, the array will be allocated inside

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.11 ilmErrorTypes ilm_getSurfaceIDsOnLayer (t_ilm_layer *layer*, t_ilm_int * *pLength*, t_ilm_surface ** *ppArray*)

Get all SurfaceIds which are currently registered to a given layer and are managed by the services.

Parameters

in	*layer*	Id of the Layer whose surfaces are to be returned
out	*pLength*	Pointer where the array length of ids should be stored
out	*ppArray*	Array where the surface id should be stored, the array will be allocated inside

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.12 ilmErrorTypes ilm_layerAddNotification (t_ilm_layer *layer*, layerNotificationFunc *callback*)

register for notification on property changes of layer

Parameters

in	*layer*	id of layer to register for notification
in	*callback*	pointer to function to be called for notification

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.
ILM_ERROR_INVALID_ARGUMENT if the given layer already has notification callback registered

21.4.4.13 ilmErrorTypes ilm_layerCreateWithDimension (t_ilm_layer * *pLayerId*, t_ilm_uint *width*, t_ilm_uint *height*)

Create a layer which should be managed by the service.

Parameters

out	*pLayerId*	pointer where the id should be/is stored. It is possible to set a id from outside, 0 will create a new id.
in	*width*	horizontal dimension of the layer
in	*height*	vertical dimension of the layer

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.14 ilmErrorTypes ilm_layerGetCapabilities (t_ilm_layer *layerId,* t_ilm_layercapabilities * *pCapabilities*)

Get the capabilities of a layer.

Parameters

in	*layerId*	Id of the layer to obtain the capabilities of
out	*pCapabilities*	The address where the capabilites are returned

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.15 ilmErrorTypes ilm_layerGetDimension (t_ilm_layer *layerId,* t_ilm_uint * *pDimension*)

Get the horizontal and vertical dimension of the layer.

Parameters

in	*layerId*	Id of layer.
out	*pDimension*	pointer to an array where the dimension should be stored. dimension[0]=width, dimension[1]=height

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.16 ilmErrorTypes ilm_layerGetOpacity (t_ilm_layer *layerId,* t_ilm_float * *pOpacity*)

Get the opacity of a layer.

Parameters

in	*layerId*	Id of the layer to obtain the opacity of.
out	*pOpacity*	pointer where the layer opacity should be stored. 0.0 means the layer is fully transparent, 1.0 means the layer is fully opaque

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.17 ilmErrorTypes ilm_layerGetOrientation (t_ilm_layer *layerId,* ilmOrientation * *pOrientation*)

Gets the orientation of a layer.

Parameters

in	*layerId*	Id of layer.
out	*pOrientation*	Address where orientation of the layer should be stored.

Note

ilmOrientation for more information on orientation values

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.18 ilmErrorTypes ilm_layerGetPosition (t_ilm_layer *layerId*, t_ilm_uint * *pPosition*)

Get the horizontal and vertical position of the layer.

Parameters

in	*layerId*	Id of layer.
out	*pPosition*	pointer to an array where the position should be stored. dimension[0]=width, dimension[1]=height

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.19 ilmErrorTypes ilm_layerGetType (t_ilm_layer *layerId*, ilmLayerType * *pLayerType*)

Get the current type of the layer.

Parameters

in	*layerId*	Id of the layer.
out	*pLayerType*	pointer to the layerType where the result should be stored.

Note

ilmLayerType for information on supported types

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.20 ilmErrorTypes ilm_layerGetVisibility (t_ilm_layer *layerId*, t_ilm_bool * *pVisibility*)

Get the visibility of a layer. If a layer is not visible, the layer and its surfaces will not be rendered.

Parameters

in	*layerId*	Id of layer.
out	*pVisibility*	pointer where the visibility of the layer should be stored ILM_SUCCESS if the Layer is visible, ILM_FALSE if the visibility is disabled.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.21 ilmErrorTypes ilm_layerRemove (t_ilm_layer *layerId*)

Removes a layer which is currently managed by the service.

Parameters

in	*layerId*	Layer to be removed

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.22 ilmErrorTypes ilm_layerRemoveNotification (t_ilm_layer *layer*)

remove notification on property changes of layer

Parameters

in	*layer*	id of layer to remove notification

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.
ILM_ERROR_INVALID_ARGUMENT if the given layer has no notification callback registered

21.4.4.23 ilmErrorTypes ilm_layerSetChromaKey (t_ilm_layer *layerId,* t_ilm_int * *pColor*)

Sets the color value which defines the transparency value.

Parameters

in	*layerId*	Id of layer.
in	*pColor*	array of the color value which is defined in red,green, blue

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.24 ilmErrorTypes ilm_layerSetDestinationRectangle (t_ilm_layer *layerId,* t_ilm_int *x,* t_ilm_int *y,* t_ilm_int *width,* t_ilm_int *height*)

Set the destination area on the display for a layer. The layer will be scaled and positioned to this rectangle for rendering.

Parameters

in	*layerId*	Id of the layer.
in	*x*	horizontal start position of the used area
in	*y*	vertical start position of the used area
in	*width*	width of the area
in	*height*	height of the area

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.25 ilmErrorTypes ilm_layerSetDimension (t_ilm_layer *layerId,* t_ilm_uint * *pDimension*)

Set the horizontal and vertical dimension of the layer.

Parameters

in	*layerId*	Id of layer.
in	*pDimension*	pointer to an array where the dimension is stored. dimension[0]=width, dimension[1]=height

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.26 ilmErrorTypes ilm_layerSetOpacity (t_ilm_layer *layerId,* t_ilm_float *opacity*)

Set the opacity of a layer.

Parameters

in	*layerId*	Id of the layer.
in	*opacity*	0.0 means the layer is fully transparent, 1.0 means the layer is fully opaque

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.27 ilmErrorTypes ilm_layerSetOrientation (t_ilm_layer *layerId,* ilmOrientation *orientation*)

Sets the orientation of a layer.

Parameters

in	*layerId*	Id of layer.
in	*orientation*	Orientation of the layer.

Note

ilmOrientation for more information on orientation values

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.28 ilmErrorTypes ilm_layerSetPosition (t_ilm_layer *layerId,* t_ilm_uint * *pPosition*)

Sets the horizontal and vertical position of the layer.

Parameters

in	*layerId*	Id of layer.
in	*pPosition*	pointer to an array where the position is stored. dimension[0]=x, dimension[1]=y

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.29 ilmErrorTypes ilm_layerSetRenderOrder (t_ilm_layer *layerId,* t_ilm_layer * *pSurfaceId,* t_ilm_int *number*)

Sets render order of surfaces within one layer.

Parameters

in	*layerId*	Id of layer.
in	*pSurfaceId*	array of surface ids
in	*number*	Number of elements in the given array of ids

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.30 ilmErrorTypes ilm_layerSetSourceRectangle (t_ilm_layer *layerId,* t_ilm_uint *x,* t_ilm_uint *y,* t_ilm_uint *width,* t_ilm_uint *height*)

Set the area of a layer which should be used for the rendering. Only this part will be visible.

Parameters

in	*layerId*	Id of the layer.
in	*x*	horizontal start position of the used area
in	*y*	vertical start position of the used area
in	*width*	width of the area
in	*height*	height of the area

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.31 ilmErrorTypes ilm_layerSetVisibility (t_ilm_layer *layerId,* t_ilm_bool *newVisibility*)

Set the visibility of a layer. If a layer is not visible, the layer and its surfaces will not be rendered.

Parameters

in	*layerId*	Id of the layer.
in	*newVisibility*	ILM_SUCCESS sets layer visible, ILM_FALSE disables the visibility.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.32 ilmErrorTypes ilm_layerTypeGetCapabilities (ilmLayerType *layerType,* t_ilm_layercapabilities * *pCapabilities*)

Get the possible capabilities of a layertype.

Parameters

in	*layerType*	The layertype to obtain the capabilities of
out	*pCapabilities*	The address where the capabilites are returned

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.33 ilmErrorTypes ilm_SetKeyboardFocusOn (t_ilm_surface *surfaceId*)

Set the keyboard focus on a certain surface To receive keyboard events, 2 conditions must be fulfilled: 1- The surface must accept events from keyboard. See ilm_UpdateInputEventAcceptanceOn 2- The keyboard focus must be set on that surface.

Parameters

in	*surfaceId*	Identifier of the surface to set the keyboard focus on.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.34 ilmErrorTypes ilm_SetOptimizationMode (ilmOptimization *id,* ilmOptimizationMode *mode*)

Enable or disable a rendering optimization.

Parameters

in	*id*	which optimization to change
in	*mode*	the mode to set on the optimzation (e.g. ON, OFF, AUTO)

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.35 ilmErrorTypes ilm_surfaceGetOpacity (const t_ilm_surface *surfaceId*, t_ilm_float * *pOpacity*)

Get the opacity of a surface.

Parameters

in	*surfaceId*	Id of the surface to get the opacity of.
out	*pOpacity*	pointer where the surface opacity should be stored. 0.0 means the surface is fully transparent, 1.0 means the surface is fully opaque

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.36 ilmErrorTypes ilm_surfaceGetOrientation (t_ilm_surface *surfaceId*, ilmOrientation * *pOrientation*)

Gets the orientation of a surface.

Parameters

in	*surfaceId*	Id of surface.
out	*pOrientation*	Address where orientation of the surface should be stored.

Note

ilmOrientation for information about orientation values

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.37 ilmErrorTypes ilm_surfaceGetPixelformat (t_ilm_layer *surfaceId*, ilmPixelFormat * *pPixelformat*)

Gets the pixelformat of a surface.

Parameters

in	*surfaceId*	Id of surface.
out	*pPixelformat*	Pointer where the pixelformat of the surface should be stored

Note

ilmPixelFormat for information about pixel format values

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.38 ilmErrorTypes ilm_surfaceGetPosition (t_ilm_surface *surfaceId,* t_ilm_uint * *pPosition*)

Get the horizontal and vertical position of the surface.

Parameters

in	*surfaceId*	Id of surface.
out	*pPosition*	pointer to an array where the position should be stored. position[0]=x, position[1]=y

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.39 ilmErrorTypes ilm_surfaceInitialize (t_ilm_surface * *pSurfaceId*)

Create the logical surface, which has no native buffer associated.

Parameters

in	*pSurfaceId*	The value pSurfaceId points to is used as ID for new surface;
out	*pSurfaceId*	The ID of the newly created surface is returned in this parameter

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.40 ilmErrorTypes ilm_surfaceSetChromaKey (t_ilm_surface *surfaceId,* t_ilm_int * *pColor*)

Sets the color value which defines the transparency value of a surface.

Parameters

in	*surfaceId*	Id of the surface to set the chromakey of.
in	*pColor*	array of the color value which is defined in red, green, blue

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.41 ilmErrorTypes ilm_surfaceSetDestinationRectangle (t_ilm_surface *surfaceId,* t_ilm_int *x,* t_ilm_int *y,* t_ilm_int *width,* t_ilm_int *height*)

Set the destination area of a surface within a layer for rendering. The surface will be scaled to this rectangle for rendering.

Parameters

in	*surfaceId*	Id of surface.
in	*x*	horizontal start position of the used area
in	*y*	vertical start position of the used area
in	*width*	width of the area
in	*height*	height of the area

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.42 ilmErrorTypes ilm_surfaceSetDimension (t_ilm_surface *surfaceId,* t_ilm_uint * *pDimension*)

Set the horizontal and vertical dimension of the surface.

Parameters

in	*surfaceId*	Id of surface.
in	*pDimension*	pointer to an array where the dimension is stored. dimension[0]=width, dimension[1]=height

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.43 ilmErrorTypes ilm_surfaceSetOpacity (const t_ilm_surface *surfaceId,* t_ilm_float *opacity*)

Set the opacity of a surface.

Parameters

surfaceId	Id of the surface to set the opacity of.
opacity	0.0 means the surface is fully transparent, 1.0 means the surface is fully opaque

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.44 ilmErrorTypes ilm_surfaceSetOrientation (t_ilm_surface *surfaceId,* ilmOrientation *orientation*)

Sets the orientation of a surface.

Parameters

in	*surfaceId*	Id of surface.
in	*orientation*	Orientation of the surface.

Note

ilmOrientation for information about orientation values

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.45 ilmErrorTypes ilm_surfaceSetPosition (t_ilm_surface *surfaceId,* t_ilm_uint * *pPosition*)

Sets the horizontal and vertical position of the surface.

Parameters

in	*surfaceId*	Id of surface.
in	*pPosition*	pointer to an array where the position is stored. position[0]=x, position[1]=y

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.46 ilmErrorTypes ilm_surfaceSetVisibility (t_ilm_surface *surfaceId,* t_ilm_bool *newVisibility*)

Set the visibility of a surface. If a surface is not visible it will not be rendered.

Parameters

in	*surfaceId*	Id of the surface to set the visibility of
in	*newVisibility*	ILM_SUCCESS sets surface visible, ILM_FALSE disables the visibility.

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.47 ilmErrorTypes ilm_takeLayerScreenshot (t_ilm_const_string *filename,* t_ilm_layer *layerid*)

Take a screenshot of a certain layer The screenshot is saved as bmp file with the corresponding filename.

Parameters

in	*filename*	Location where the screenshot should be stored
in	*layerid*	Identifier of the layer to take the screenshot of

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.48 ilmErrorTypes ilm_takeScreenshot (t_ilm_uint *screen*, t_ilm_const_string *filename*)

Take a screenshot from the current displayed layer scene. The screenshot is saved as bmp file with the corresponding filename.

Parameters

in	*screen*	Id of screen where screenshot should be taken
in	*filename*	Location where the screenshot should be stored

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.4.4.49 ilmErrorTypes ilm_takeSurfaceScreenshot (t_ilm_const_string *filename*, t_ilm_surface *surfaceid*)

Take a screenshot of a certain surface The screenshot is saved as bmp file with the corresponding filename.

Parameters

in	*filename*	Location where the screenshot should be stored
in	*surfaceid*	Identifier of the surface to take the screenshot of

Returns

ILM_SUCCESS if the method call was successful
ILM_FAILED if the client can not call the method on the service.

21.5 Layer Management Scene API

Functions

- virtual IScene::~IScene ()

  default destructor
- virtual Layer * IScene::createLayer (const uint id, int creatorPid)=0

  Creates a new layer within the scene.
- virtual Surface * IScene::createSurface (const uint id, int creatorPid)=0

  Creates a new surface within the scene.
- virtual bool IScene::removeLayer (Layer *layer)=0

  Remove a layer from the scene.
- virtual bool IScene::removeSurface (Surface *surface)=0

  Remove surface from scene.
- virtual LmScreen * IScene::getScreen (const uint id) const =0

  Get a screen of the scene by id.
- virtual Layer * IScene::getLayer (const uint id)=0

  Get a layer of the scene by id.
- virtual Surface * IScene::getSurface (const uint id)=0

  Get a surface of the scene by id.
- virtual void IScene::getLayerIDs (uint *length, uint **array) const =0

  Get list of ids of all layers currently existing.
- virtual bool IScene::getLayerIDsOfScreen (const uint screenID, uint *length, uint **array) const =0

  Get list of ids of all layers currently existing.
- virtual void IScene::getSurfaceIDs (uint *length, uint **array) const =0

  Get list of ids of all surfaces currently existing.
- virtual void IScene::lockScene ()=0

  Lock the list for read and write access.
- virtual void IScene::unlockScene ()=0

  Unlock the list for read and write access.
- virtual LayerList & IScene::getCurrentRenderOrder (const uint id)=0

  Get the current render order of the scene.
- virtual LmScreenList & IScene::getScreenList ()=0

  Get the screen list of the scene.
- virtual const SurfaceMap IScene::getAllSurfaces () const =0

  Get a map of all surface from the scene.
- virtual bool IScene::isLayerInCurrentRenderOrder (const uint id)=0

  Check, if layer is in render order.

21.5.1 Detailed Description

21.5.2 Function Documentation

21.5.2.1 virtual Layer* IScene::createLayer (const uint *id*, int *creatorPid*) [pure virtual]

Creates a new layer within the scene.

Parameters

in	*id*	id of layer
in	*creatorPid*	client process id that requested the creation of this layer

Returns

pointer to layer

Implemented in Scene.

21.5.2.2 virtual Surface* IScene::createSurface (const uint *id*, int *creatorPid*) [pure virtual]

Creates a new surface within the scene.

Parameters

in	*id*	id of surface
in	*creatorPid*	client process id that requested the creation of this surface

Returns

pointer to surface

Implemented in Scene.

21.5.2.3 virtual const SurfaceMap IScene::getAllSurfaces ( ) const [pure virtual]

Get a map of all surface from the scene.

Returns

Map holding all surfaces.

Implemented in Scene.

21.5.2.4 virtual LayerList& IScene::getCurrentRenderOrder (const uint *id*) [pure virtual]

Get the current render order of the scene.

Parameters

in	*id*	screen id

Returns

reference to render order

Implemented in Scene.

21.5.2.5 virtual Layer* IScene::getLayer (const uint *id*) [pure virtual]

Get a layer of the scene by id.

Parameters

in	*id*	id of the layer

Returns

pointer to the layer with id

Implemented in Scene.

21.5.2.6 virtual void IScene::getLayerIDs (uint * *length*, uint ** *array*) const [pure virtual]

Get list of ids of all layers currently existing.

Parameters

out	*length*	length of array returned in array
out	*array*	array containing the ids of all layers

Returns

list of ids of all currently know layers

Implemented in Scene.

21.5.2.7 virtual bool IScene::getLayerIDsOfScreen (const uint *screenID*, uint * *length*, uint ** *array*) const [pure virtual]

Get list of ids of all layers currently existing.

Parameters

in	*screenID*	id of screen
out	*length*	length of array returned in array
out	*array*	array containing the ids of all layers on screen

Returns

list of ids of all currently know layers

Implemented in Scene.

21.5.2.8 virtual LmScreen* IScene::getScreen (const uint *id*) const [pure virtual]

Get a screen of the scene by id.

Parameters

in	*id*	id of the screen

Returns

pointer to the screen with id

Implemented in Scene.

21.5.2.9 virtual LmScreenList& IScene::getScreenList ( ) [pure virtual]

Get the screen list of the scene.

Returns

reference to screen list

Implemented in Scene.

21.5.2.10 virtual Surface* IScene::getSurface (const uint *id*) [pure virtual]

Get a surface of the scene by id.

Parameters

in	*id*	id of the surface

Returns

pointer to the surface with id

Implemented in Scene.

21.5.2.11 virtual void IScene::getSurfaceIDs (uint * *length*, uint ** *array*) const [pure virtual]

Get list of ids of all surfaces currently existing.

Parameters

out	*length*	length of array returned in array
out	*array*	array containing the ids of all surfaces

Returns

list of ids of all currently know surfaces

Implemented in Scene.

21.5.2.12 virtual bool IScene::isLayerInCurrentRenderOrder (const uint *id*) [pure virtual]

Check, if layer is in render order.

Parameters

in	*id*	layer id

Returns

TRUE: layer is in render order
FALSE: layer is not in render order

Implemented in Scene.

21.5.2.13 virtual void IScene::lockScene ( ) [pure virtual]

Lock the list for read and write access.

Implemented in Scene.

21.5.2.14 virtual bool IScene::removeLayer (Layer * *layer*) [pure virtual]

Remove a layer from the scene.

Parameters

in	*layer*	pointer to layer

Implemented in Scene.

21.5.2.15 virtual bool IScene::removeSurface (Surface * *surface*) [pure virtual]

Remove surface from scene.

Parameters

in	*surface*	pointer to surface

Implemented in Scene.

21.5.2.16 virtual void IScene::unlockScene ( ) [pure virtual]

Unlock the list for read and write access.

Implemented in Scene.

21.5.2.17 virtual IScene::~IScene ( ) [inline], [virtual]

default destructor

21.6 Layer Management Renderer API

Functions

- virtual bool IRenderer::start (int width, int height, const char *displayName)=0

  Start the actual rendering process (render loop)
- virtual void IRenderer::stop ()=0

  Stop rendering process (stop render loop)
- virtual void IRenderer::setdebug (bool onoff)=0

  Switch debug mode of this component on or off.
- virtual void IRenderer::doScreenShot (std::string fileToSave)=0

  Store graphical content of screen to bitmap.
- virtual void IRenderer::doScreenShotOfLayer (std::string fileToSave, const unsigned int id)=0

  Store graphical content of layer to bitmap.
- virtual void IRenderer::doScreenShotOfSurface (std::string fileToSave, const unsigned int id, const unsigned int layer_id)=0

  Store graphical content of surface to bitmap.
- virtual unsigned int IRenderer::getLayerTypeCapabilities (LayerType layertype)=0

  Get the capabilies of a layer type.
- virtual unsigned int IRenderer::getNumberOfHardwareLayers (unsigned int screenID)=0

  Get the number of supported hardware layers of the renderer for a screen.
- virtual unsigned int * IRenderer::getScreenResolution (unsigned int screenID)=0

  Get the resolution of a screen handled by this renderer.
- virtual unsigned int * IRenderer::getScreenIDs (unsigned int *length)=0

  Get the list if available screen ids.
- virtual Shader * IRenderer::createShader (const string *vertexName, const string *fragmentName)=0

  Create a shader object (that can be applied to surfaces)
- virtual void IRenderer::signalWindowSystemRedraw ()=0

  Trigger a redraw for this renderer.
- virtual void IRenderer::forceCompositionWindowSystem ()=0

  Force composition for entire scene.
- virtual InputManager * IRenderer::getInputManager () const =0

  Get the InputManager associated to the Scene.
- virtual bool IRenderer::setOptimizationMode (OptimizationType id, OptimizationModeType mode)=0

  Set the mode for the specified optimization (e.g. OFF,ON,AUTO)
- virtual bool IRenderer::getOptimizationMode (OptimizationType id, OptimizationModeType *mode)=0

  Get the current mode for the specified optimization.

21.6.1 Detailed Description

Abstract Base of all CompositingControllers, ie Renderers.

21.6.2 Function Documentation

21.6.2.1 virtual Shader* IRenderer::createShader (const string * *vertexName*, const string * *fragmentName*) [pure virtual]

Create a shader object (that can be applied to surfaces)

Parameters

in	*vertexName*	filename of vertex shader source code
in	*fragmentName*	filename of fragment shader source code

Returns

Pointer to created shader object

21.6.2.2 virtual void IRenderer::doScreenShot (std::string *fileToSave*) [pure virtual]

Store graphical content of screen to bitmap.

Parameters

in	*fileToSave*	path to bitmap file to store the graphical content

21.6.2.3 virtual void IRenderer::doScreenShotOfLayer (std::string *fileToSave*, const unsigned int *id*) [pure virtual]

Store graphical content of layer to bitmap.

Parameters

in	*fileToSave*	path to bitmap file to store the graphical content
in	*id*	id of layer

21.6.2.4 virtual void IRenderer::doScreenShotOfSurface (std::string *fileToSave*, const unsigned int *id*, const unsigned int *layer_id*) [pure virtual]

Store graphical content of surface to bitmap.

Parameters

in	*fileToSave*	path to bitmap file to store the graphical content
in	*id*	id of surface
in	*layer_id*	id of layer

21.6.2.5 virtual void IRenderer::forceCompositionWindowSystem ( ) [pure virtual]

Force composition for entire scene.

21.6.2.6 virtual InputManager* IRenderer::getInputManager ( ) const [pure virtual]

Get the InputManager associated to the Scene.

21.6.2.7 virtual unsigned int IRenderer::getLayerTypeCapabilities (LayerType *layertype*) [pure virtual]

Get the capabilies of a layer type.

Parameters

in	*layertype*	type of layer

Returns

bitset with flags set for capabilities

21.6.2.8 virtual unsigned int IRenderer::getNumberOfHardwareLayers (unsigned int *screenID*) [pure virtual]

Get the number of supported hardware layers of the renderer for a screen.

Parameters

in	*screenID*	id of the screen

Returns

Number of supported hardware layers for screen

21.6.2.9 virtual bool IRenderer::getOptimizationMode (OptimizationType *id,* OptimizationModeType * *mode*) [pure virtual]

Get the current mode for the specified optimization.

Parameters

in	*id*	id of optimization
out	*mode*	retrieved mode value

Returns

TRUE: id is valid and mode was returned
FALSE: id was invalid and/or mode was not returned

21.6.2.10 virtual unsigned int* IRenderer::getScreenIDs (unsigned int * *length*) [pure virtual]

Get the list if available screen ids.

Parameters

out	*length*	length of the returned array

Returns

array containing all available screen ids

21.6.2.11 virtual unsigned int* IRenderer::getScreenResolution (unsigned int *screenID*) [pure virtual]

Get the resolution of a screen handled by this renderer.

Parameters

in	*screenID*	id of the screen

Returns

array with width and height of screen

21.6.2.12 virtual void IRenderer::setdebug (bool *onoff*) [pure virtual]

Switch debug mode of this component on or off.

Parameters

in	*onoff*	TRUE: Turn on debug mode, FALSE: Turn off debug mode

21.6.2.13 virtual bool IRenderer::setOptimizationMode (OptimizationType *id,* OptimizationModeType *mode*) [pure virtual]

Set the mode for the specified optimization (e.g. OFF,ON,AUTO)

Parameters

in	*id*	id of optimization
in	*mode*	mode to set for the optimization

Returns

TRUE: id and mode are valid and mode was set
FALSE: id or mode was invalid and/or mode could not be set

21.6.2.14 virtual void IRenderer::signalWindowSystemRedraw ( ) [pure virtual]

Trigger a redraw for this renderer.

21.6.2.15 virtual bool IRenderer::start (int *width,* int *height,* const char * *displayName*) [pure virtual]

Start the actual rendering process (render loop)

Parameters

in	*width*	width of display handled by this renderer
in	*height*	height of display handled by this renderer
in	*displayName*	name of display handled by this renderer

Returns

TRUE: renderer was started successfully
FALSE: renderer start failed

21.6.2.16 virtual void IRenderer::stop ( ) [pure virtual]

Stop rendering process (stop render loop)

21.7 Layer Management Communicator API

Functions

- ICommunicator::ICommunicator (ICommandExecutor *executor)

  constructor: any communicator need a executor for commands
- virtual bool ICommunicator::start ()=0

  Start communication process, i.e. start specific listening process of communication method.
- virtual void ICommunicator::stop ()=0

  Stop communication. Stop sending command objects.
- virtual void ICommunicator::process (int timeout_ms)=0

  Process communication.
- virtual void ICommunicator::setdebug (bool onoff)=0

  Switch debug mode of this component on or off.

21.7.1 Detailed Description

Abstract Base of all Communicator plugins.

21.7.2 Function Documentation

21.7.2.1 ICommunicator::ICommunicator (ICommandExecutor * *executor*) [inline]

constructor: any communicator need a executor for commands

Parameters

in	*executor*	Pointer to an object to send commands to

21.7.2.2 virtual void ICommunicator::process (int *timeout_ms*) [pure virtual]

Process communication.

Parameters

in	*timeout_ms*	timeout value in milliseconds

21.7.2.3 virtual void ICommunicator::setdebug (bool *onoff*) [pure virtual]

Switch debug mode of this component on or off.

Parameters

in	*onoff*	TRUE: Turn on debug mode, FALSE: Turn off debug mode

21.7.2.4 virtual bool ICommunicator::start ( ) [pure virtual]

Start communication process, i.e. start specific listening process of communication method.

21.7.2.5 virtual void ICommunicator::stop ( ) [pure virtual]

Stop communication. Stop sending command objects.

21.8 Layer Management Commands

Functions

- CommitCommand::CommitCommand (pid_t sender)
- DebugCommand::DebugCommand (pid_t sender, bool onoff)
- ExitCommand::ExitCommand (pid_t sender)
- GetOptimizationModeCommand::GetOptimizationModeCommand (pid_t sender, OptimizationType id, OptimizationModeType ∗returnMode)
- LayerAddSurfaceCommand::LayerAddSurfaceCommand (pid_t sender, unsigned int layerid, unsigned int surfaceid)
- LayerCreateCommand::LayerCreateCommand (pid_t sender, uint OriginalWidth, uint OriginalHeight, uint ∗idReturn)
- LayerDumpCommand::LayerDumpCommand (pid_t sender, char ∗givenfilename, unsigned int id=0)
- LayerGetDimensionCommand::LayerGetDimensionCommand (pid_t sender, int id, unsigned int ∗widthRet, unsigned int ∗heightRet)
- LayerGetOpacityCommand::LayerGetOpacityCommand (pid_t sender, int id, double ∗returnOpacity)
- LayerGetOrientationCommand::LayerGetOrientationCommand (pid_t sender, int id, OrientationType ∗orientation)
- LayerGetPositionCommand::LayerGetPositionCommand (pid_t sender, int id, unsigned int ∗xRet, unsigned int ∗yRet)
- LayerGetVisibilityCommand::LayerGetVisibilityCommand (pid_t sender, int id, bool ∗visibility)
- LayerRemoveCommand::LayerRemoveCommand (pid_t sender, unsigned int objectID)
- LayerRemoveSurfaceCommand::LayerRemoveSurfaceCommand (pid_t sender, unsigned layerid, unsigned surfaceid)
- LayerSetChromaKeyCommand::LayerSetChromaKeyCommand (pid_t sender, unsigned int layerid, unsigned int ∗array, unsigned int length)
- LayerSetDestinationRectangleCommand::LayerSetDestinationRectangleCommand (pid_t sender, int id, unsigned int x, unsigned int y, unsigned int width, unsigned int height)
- LayerSetDimensionCommand::LayerSetDimensionCommand (pid_t sender, int id, unsigned int width, unsigned int height)
- LayerSetOpacityCommand::LayerSetOpacityCommand (pid_t sender, unsigned int id, double Opacity)
- LayerSetOrientationCommand::LayerSetOrientationCommand (pid_t sender, unsigned int id, OrientationType Orientation)
- LayerSetPositionCommand::LayerSetPositionCommand (pid_t sender, unsigned int id, unsigned int x, unsigned int y)
- LayerSetRenderOrderCommand::LayerSetRenderOrderCommand (pid_t sender, unsigned int layerid, unsigned int ∗array, unsigned int length)
- LayerSetSourceRectangleCommand::LayerSetSourceRectangleCommand (pid_t sender, int id, unsigned int x, unsigned int y, unsigned int width, unsigned int height)
- LayerSetTypeCommand::LayerSetTypeCommand (pid_t sender, const unsigned int givenid, LayerType layertype)
- LayerSetVisibilityCommand::LayerSetVisibilityCommand (pid_t sender, const unsigned int givenid, bool newvisibility)
- ScreenDumpCommand::ScreenDumpCommand (pid_t sender, char ∗givenfilename, unsigned int id=0)
- ScreenSetRenderOrderCommand::ScreenSetRenderOrderCommand (pid_t sender, unsigned int screenID, unsigned int ∗array, unsigned int length)
- SetOptimizationModeCommand::SetOptimizationModeCommand (pid_t sender, OptimizationType id, OptimizationModeType mode)

- ShaderCreateCommand::ShaderCreateCommand (pid_t sender, const std::string &vertName, const std::string &fragName, unsigned int *id)
- ShaderDestroyCommand::ShaderDestroyCommand (pid_t sender, unsigned int shaderid)
- ShaderSetUniformsCommand::ShaderSetUniformsCommand (pid_t sender, unsigned int shaderid, const std::vector< std::string > &uniforms)
- SurfaceCreateCommand::SurfaceCreateCommand (pid_t sender, uint *idReturn)
- SurfaceDumpCommand::SurfaceDumpCommand (pid_t sender, char *givenfilename, unsigned int id=0)
- SurfaceGetDimensionCommand::SurfaceGetDimensionCommand (pid_t sender, int id, unsigned int *widthRet, unsigned int *heightRet)
- SurfaceGetKeyboardFocusCommand::SurfaceGetKeyboardFocusCommand (pid_t sender, unsigned int *pSurfId)
- SurfaceGetOpacityCommand::SurfaceGetOpacityCommand (pid_t sender, int id, double *returnOpacity)
- SurfaceGetOrientationCommand::SurfaceGetOrientationCommand (pid_t sender, int id, OrientationType *orientation)
- SurfaceGetPixelformatCommand::SurfaceGetPixelformatCommand (pid_t sender, int id, PixelFormat *f)
- SurfaceGetPositionCommand::SurfaceGetPositionCommand (pid_t sender, int id, unsigned int *xRet, unsigned int *yRet)
- SurfaceGetVisibilityCommand::SurfaceGetVisibilityCommand (pid_t sender, int id, bool *visibility)
- SurfaceRemoveCommand::SurfaceRemoveCommand (pid_t sender, unsigned int objectID)
- SurfaceRemoveNativeContentCommand::SurfaceRemoveNativeContentCommand (pid_t sender, unsigned int surfaceId)
- SurfaceSetChromaKeyCommand::SurfaceSetChromaKeyCommand (pid_t sender, unsigned int surfaceid, unsigned int *array, unsigned int length)
- SurfaceSetDestinationRectangleCommand::SurfaceSetDestinationRectangleCommand (pid_t sender, int id, unsigned int x, unsigned int y, unsigned int width, unsigned int height)
- SurfaceSetDimensionCommand::SurfaceSetDimensionCommand (pid_t sender, int id, unsigned int width, unsigned int height)
- SurfaceSetKeyboardFocusCommand::SurfaceSetKeyboardFocusCommand (pid_t sender, unsigned int surfId)
- SurfaceSetNativeContentCommand::SurfaceSetNativeContentCommand (pid_t sender, unsigned int surfaceId, unsigned int handle, PixelFormat pixelformat, uint OriginalWidth, uint OriginalHeight)
- SurfaceSetOpacityCommand::SurfaceSetOpacityCommand (pid_t sender, unsigned int id, double Opacity)
- SurfaceSetOrientationCommand::SurfaceSetOrientationCommand (pid_t sender, unsigned int id, OrientationType Orientation)
- SurfaceSetPositionCommand::SurfaceSetPositionCommand (pid_t sender, unsigned int id, unsigned int x, unsigned int y)
- SurfaceSetShaderCommand::SurfaceSetShaderCommand (pid_t sender, unsigned int id, unsigned int shaderid)
- SurfaceSetSourceRectangleCommand::SurfaceSetSourceRectangleCommand (pid_t sender, int id, unsigned int x, unsigned int y, unsigned int width, unsigned int height)
- SurfaceSetVisibilityCommand::SurfaceSetVisibilityCommand (pid_t sender, const unsigned int givenid, bool newvisibility)
- SurfaceUpdateInputEventAcceptance::SurfaceUpdateInputEventAcceptance (pid_t sender, unsigned int surfId, InputDevice devices, bool accept)

21.8.1 Detailed Description

21.8.2 Function Documentation

21.8.2.1 CommitCommand::CommitCommand (pid_t *sender*) [inline]

Action Taken This command executes all enqueued asynchronous commands within the GENIVI Layer-Management

Expected Frequency Called after one or more calls of changing properties.

Parameters

in	*sender*	process id of application that sent this command

21.8.2.2 DebugCommand::DebugCommand (pid_t *sender,* bool *onoff*) [inline]

Action Taken This command sets the debug mode within the GENIVI LayerManagement

Expected Frequency Used only for development and debugging.

Parameters

in	*sender*	process id of application that sent this command
in	*onoff*	TRUE: enable debug mode, FALSE: disable debug mode

21.8.2.3 ExitCommand::ExitCommand (pid_t *sender*) [inline]

Action Taken This command triggers a service shutdown within the GENIVI LayerManagement

Expected Frequency Can be called only once.

Parameters

in	*sender*	process id of application that sent this command

21.8.2.4 GetOptimizationModeCommand::GetOptimizationModeCommand (pid_t *sender,* OptimizationType *id,* OptimizationModeType * *returnMode*) [inline]

Action Taken This command returns the current mode for the specified optimization.

Parameters

in	*sender*	client process id that sent this command
in	*id*	id of optimization
in	*returnMode*	location to store mode of optimization on execution

21.8.2.5 LayerAddSurfaceCommand::LayerAddSurfaceCommand (pid_t *sender,* unsigned int *layerid,* unsigned int *surfaceid*) [inline]

Action Taken This command adds a surface to a layer within the GENIVI LayerManagement

Expected Frequency Typically surfaces will be added to one or more layers once in their life cycle. So this will typically be called at least once for every surface created.

Parameters

in	*sender*	process id of application that sent this command
in	*layerid*	id of layer
in	*surfaceid*	id of surface

21.8.2.6 LayerCreateCommand::LayerCreateCommand (pid_t *sender,* uint *OriginalWidth,* uint *OriginalHeight,* uint * *idReturn*) [inline]

Action Taken This command creates a new layer within the GENIVI LayerManagement

Expected Frequency The output of several applications is grouped into layers so they can be adjusted together. This means there will be less layers than surfaces. A small configuration might create a layer for everything concerning OEM branding, one layer for third party applications and one layer for status applications.

Parameters

in	*sender*	process id of application that sent this command
in	*OriginalWidth*	width of the original layer
in	*OriginalHeight*	height of the original layer
in	*idReturn*	location to store layer id on execution; pre-initialized value will be used as new id for the layer to be created

21.8.2.7 LayerDumpCommand::LayerDumpCommand (pid_t *sender,* char * *givenfilename,* unsigned int *id* = 0) [inline]

Action Taken This command stores a bitmap file with the graphical content of a layer within the GENIVI LayerManagement

Expected Frequency Used for layer management.

Parameters

in	*sender*	process id of application that sent this command
in	*givenfilename*	path and filename to store bitmap file
in	*id*	

21.8.2.8 LayerGetDimensionCommand::LayerGetDimensionCommand (pid_t *sender,* int *id,* unsigned int * *widthRet,* unsigned int * *heightRet*) [inline]

Action Taken This command returns the dimensions of a layer within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*widthRet*	location to return width of layer on execution
in	*heightRet*	location to return height of layer on execution

21.8.2.9 LayerGetOpacityCommand::LayerGetOpacityCommand (pid_t *sender,* int *id,* double * *returnOpacity*) [inline]

Action Taken This command returns the opacity of a layer within the GENIVI LayerManagement

Expected Frequency Can occur very frequently for animations.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*returnOpacity*	location to store opacity of layer on execution

21.8.2.10 LayerGetOrientationCommand::LayerGetOrientationCommand (pid_t *sender,* int *id,* OrientationType * *orientation*) [inline]

Action Taken This command returns the orientation of a layer within the GENIVI LayerManagement

Expected Frequency Can be used for rearrangement.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*orientation*	location to store orientation of layer on execution

21.8.2.11 LayerGetPositionCommand::LayerGetPositionCommand (pid_t *sender,* int *id,* unsigned int * *xRet,* unsigned int * *yRet*) [inline]

Action Taken This command returns the position of a layer within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*xRet*	location to return x position of layer on execution
in	*yRet*	location to return y position of layer on execution

21.8.2.12 LayerGetVisibilityCommand::LayerGetVisibilityCommand (pid_t *sender,* int *id,* bool * *visibility*) [inline]

Action Taken This command returns the visibility of a layer within the GENIVI LayerManagement

Expected Frequency Frequently when events occur within the system which cause a rearrangement of graphics, applications or contexts.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*visibility*	location to store visibility on execution

21.8.2.13 LayerRemoveCommand::LayerRemoveCommand (pid_t *sender,* unsigned int *objectID*) [inline]

Action Taken This command removes a layer within the GENIVI LayerManagement

Expected Frequency The output of several applications is grouped into layers so they can be adjusted together. This means there will be less layers than surfaces. A small configuration might create a layer for everything concerning OEM branding, one layer for third party applications and one layer for status applications.

Parameters

in	*sender*	process id of application that sent this command
in	*objectID*	id of layer

21.8.2.14 LayerRemoveSurfaceCommand::LayerRemoveSurfaceCommand (pid_t *sender,* unsigned *layerid,* unsigned *surfaceid*) [inline]

Action Taken This command removes a surface from a layer within the GENIVI LayerManagement

Expected Frequency Typically surfaces will be added to one or more layers once in their life cycle. So this will typically be called at least once for every surface created.

Parameters

in	*sender*	process id of application that sent this command
in	*layerid*	id of layer
in	*surfaceid*	id of surface

21.8.2.15 LayerSetChromaKeyCommand::LayerSetChromaKeyCommand (pid_t *sender,* unsigned int *layerid,* unsigned int * *array,* unsigned int *length*) [inline]

Action Taken This command sets the chroma key of a layer within the GENIVI LayerManagement

Expected Frequency Called in order to rearrange graphical output.

Parameters

in	*sender*	process id of application that sent this command
in	*layerid*	Id of the layer to set the chromakey of.
in	*array*	array of color value which is defined in red, green, blue
in	*length*	length of array provided as argument array

21.8.2.16 LayerSetDestinationRectangleCommand::LayerSetDestinationRectangleCommand (pid_t *sender,* int *id,* unsigned int *x,* unsigned int *y,* unsigned int *width,* unsigned int *height*) [inline]

Action Taken This command sets the destination region of a layer within the GENIVI LayerManagement

Expected Frequency Called when first initializing a new layer and for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*x*	x position of layer on screen
in	*y*	y position of layer on screen
in	*width*	width of layer on screen
in	*height*	height of layer on screen

21.8.2.17 LayerSetDimensionCommand::LayerSetDimensionCommand (pid_t *sender,* int *id,* unsigned int *width,* unsigned int *height*) [inline]

Action Taken This command sets the dimensions of a layer within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*width*	width of layer
in	*height*	height of layer

21.8.2.18 LayerSetOpacityCommand::LayerSetOpacityCommand (pid_t *sender,* unsigned int *id,* double *Opacity*) [inline]

Action Taken This command sets the opacity of a layer within the GENIVI LayerManagement

Expected Frequency Can occur very frequently for animations.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*Opacity*	opacity of layer

21.8.2.19 LayerSetOrientationCommand::LayerSetOrientationCommand (pid_t *sender,* unsigned int *id,* OrientationType *Orientation*) [inline]

Action Taken This command sets the orientation of a layer within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*Orientation*	orientation of layer

21.8.2.20 LayerSetPositionCommand::LayerSetPositionCommand (pid_t *sender,* unsigned int *id,* unsigned int *x,* unsigned int *y*) [inline]

Action Taken This command sets the position of a layer within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*x*	x position of layer on screen
in	*y*	y position of layer on screen

21.8.2.21 LayerSetRenderOrderCommand::LayerSetRenderOrderCommand (pid_t *sender,* unsigned int *layerid,* unsigned int * *array,* unsigned int *length*) [inline]

Action Taken This command sets the render order of surfaces on a layer within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*layerid*	id of layer
in	*array*	array of surface ids
in	*length*	length of array provided as argument array

21.8.2.22 LayerSetSourceRectangleCommand::LayerSetSourceRectangleCommand (pid_t *sender,* int *id,* unsigned int *x,* unsigned int *y,* unsigned int *width,* unsigned int *height*) [inline]

Action Taken This command sets the source region of a layer within the GENIVI LayerManagement

Expected Frequency Called when first initializing a new layer and for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of layer
in	*x*	x position wihtin layer
in	*y*	y position within layer
in	*width*	width within layer
in	*height*	height within layer

21.8.2.23 LayerSetTypeCommand::LayerSetTypeCommand (pid_t *sender,* const unsigned int *givenid,* LayerType *layertype*) [inline]

Action Taken This command sets the type of a layer within the GENIVI LayerManagement

Expected Frequency Called when first initializing a new layer.

Parameters

in	*sender*	process id of application that sent this command
in	*givenid*	id of layer
in	*layertype*	type of layer

21.8.2.24 LayerSetVisibilityCommand::LayerSetVisibilityCommand (pid_t *sender,* const unsigned int *givenid,* bool *newvisibility*) [inline]

Action Taken This command sets the visibility of a layer within the GENIVI LayerManagement

Expected Frequency Frequently when events occur within the system which cause a rearrangement of graphics, applications or contexts.

Parameters

in	*sender*	process id of application that sent this command
in	*givenid*	id of layer
in	*newvisibility*	TRUE: layer is set to visible, FALSE: layer is set to invisible

21.8.2.25 ScreenDumpCommand::ScreenDumpCommand (pid_t *sender,* char * *givenfilename,* unsigned int *id* = 0) [inline]

Action Taken This command stores a bitmap file with the graphical content of a screen within the GENIVI LayerManagement

Expected Frequency Used for screen management.

Parameters

in	*sender*	process id of application that sent this command
in	*givenfilename*	path and filename to store bitmap file
in	*id*	

21.8.2.26 ScreenSetRenderOrderCommand::ScreenSetRenderOrderCommand (pid_t *sender*, unsigned int *screenID*, unsigned int * *array*, unsigned int *length*) [inline]

Action Taken This command sets the render order of layers within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*screenID*	ID of screen
in	*array*	array of layer ids
in	*length*	length of array provided in parameter array

21.8.2.27 SetOptimizationModeCommand::SetOptimizationModeCommand (pid_t *sender*, OptimizationType *id*, OptimizationModeType *mode*) [inline]

Action Taken This command sets the mode for the specified optimization.

Expected Frequency Infrequent.

Parameters

in	*sender*	client process id that sent this command
in	*id*	id of optimization
in	*mode*	optimization mode to set

21.8.2.28 ShaderCreateCommand::ShaderCreateCommand (pid_t *sender*, const std::string & *vertName*, const std::string & *fragName*, unsigned int * *id*) [inline]

Action Taken This command creates a shader within the GENIVI LayerManagement

Expected Frequency Once per shader.

Parameters

in	*sender*	process id of application that sent this command
in	*vertName*	path and filename to vertex shader source file
in	*fragName*	path and filename to fragment shader source file
in	*id*	location to store shader id on execution

21.8.2.29 ShaderDestroyCommand::ShaderDestroyCommand (pid_t *sender*, unsigned int *shaderid*) [inline]

Action Taken This command destroys a shader within the GENIVI LayerManagement

Expected Frequency Once per shader.

Parameters

in	*sender*	process id of application that sent this command
in	*shaderid*	id of shader

21.8.2.30 ShaderSetUniformsCommand::ShaderSetUniformsCommand (pid_t *sender,* unsigned int *shaderid,* const std::vector< std::string > & *uniforms*) [inline]

Action Taken This command sets the uniform value of a shader within the GENIVI LayerManagement

Expected Frequency Typically for every rendered frame.

Parameters

in	*sender*	process id of application that sent this command
in	*shaderid*	id of shader
in	*uniforms*	vector holding uniforms

21.8.2.31 SurfaceCreateCommand::SurfaceCreateCommand (pid_t *sender,* uint * *idReturn*) [inline]

Action Taken This command creates a surface within the GENIVI LayerManagement

Expected Frequency Called at least once for all graphical applications, either by applications themselves or through a management component informed of a new window by the window management API.

Parameters

in	*sender*	process id of application that sent this command
in	*sender*	process id of sender of this command
in	*idReturn*	location where surface id will be stored during execution pre-initialized value is used as requested surface id

21.8.2.32 SurfaceDumpCommand::SurfaceDumpCommand (pid_t *sender,* char * *givenfilename,* unsigned int *id* = 0) [inline]

Action Taken This command stores a bitmap file with the graphical content of a surface within the GENIVI LayerManagement

Expected Frequency Used for surface management.

Parameters

in	*sender*	process id of application that sent this command
in	*givenfilename*	path and filename for bitmap file
in	*id*	id of surface

21.8.2.33 SurfaceGetDimensionCommand::SurfaceGetDimensionCommand (pid_t *sender*, int *id*, unsigned int * *widthRet*, unsigned int * *heightRet*) [inline]

Action Taken This command returns the dimension of a surface within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*widthRet*	pointer to store surface width on execution
in	*heightRet*	pointer to store surface height on execution

21.8.2.34 SurfaceGetKeyboardFocusCommand::SurfaceGetKeyboardFocusCommand (pid_t *sender*, unsigned int * *pSurfId*) [inline]

Action Taken This command returns the identifier of the surface which currently hold the keyboard focus

Expected Frequency Whenever it is needed

Parameters

in	*sender*	process id of application that sent this command
out	*pSurfId*	id of surface

21.8.2.35 SurfaceGetOpacityCommand::SurfaceGetOpacityCommand (pid_t *sender*, int *id*, double * *returnOpacity*) [inline]

Action Taken This command returns the opacity of a surface within the GENIVI LayerManagement

Expected Frequency Can be used for rearrangement.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*returnOpacity*	location to store opacity of surface on execution

21.8.2.36 SurfaceGetOrientationCommand::SurfaceGetOrientationCommand (pid_t *sender*, int *id*, OrientationType * *orientation*) [inline]

Action Taken This command returns the orientation of a surface within the GENIVI LayerManagement

Expected Frequency Can be used for rearrangement.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*orientation*	location to tore orientation of surface on execution

21.8.2.37 SurfaceGetPixelformatCommand::SurfaceGetPixelformatCommand (pid_t *sender,* int *id,* PixelFormat * *f*) [inline]

Action Taken This command returns the pixel format of a surface within the GENIVI LayerManagement

Expected Frequency

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*f*	location to store pixel format of surface on execution

21.8.2.38 SurfaceGetPositionCommand::SurfaceGetPositionCommand (pid_t *sender,* int *id,* unsigned int * *xRet,* unsigned int * *yRet*) [inline]

Action Taken This command returns the position of a surface within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*xRet*	location to store x position of surface on execution
in	*yRet*	location to store y position of surface on execution

21.8.2.39 SurfaceGetVisibilityCommand::SurfaceGetVisibilityCommand (pid_t *sender,* int *id,* bool * *visibility*) [inline]

Action Taken This command returns the visibility of a surface within the GENIVI LayerManagement

Expected Frequency Can be used for rearrangement.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*visibility*	location to store visibility on execution

21.8.2.40 SurfaceRemoveCommand::SurfaceRemoveCommand (pid_t *sender,* unsigned int *objectID*) [inline]

Action Taken This command removes a surface within the GENIVI LayerManagement

Expected Frequency Called at end of application or when an application or its window is shut down.

Parameters

in	*sender*	process id of application that sent this command
in	*objectID*	id of surface

21.8.2.41 SurfaceRemoveNativeContentCommand::SurfaceRemoveNativeContentCommand (pid_t *sender,* unsigned int *surfaceId*) [inline]

Action Taken This command removes the native content (application content) of a surface within the GENIVI LayerManagement

Expected Frequency Typically should not be needed unless a client wants to re-use the surface with multiple contents.

Parameters

in	*sender*	process id of application that sent this command
in	*surfaceId*	id of surface

21.8.2.42 SurfaceSetChromaKeyCommand::SurfaceSetChromaKeyCommand (pid_t *sender,* unsigned int *surfaceid,* unsigned int * *array,* unsigned int *length*) [inline]

Action Taken This command sets the chroma key of a surface within the GENIVI LayerManagement

Expected Frequency Called in order to rearrange graphical output.

Parameters

in	*sender*	process id of application that sent this command
in	*surfaceid*	Id of the surface to set the chromakey of.
in	*array*	array of color value which is defined in red, green, blue
in	*length*	length of array provided as argument array

21.8.2.43 SurfaceSetDestinationRectangleCommand::SurfaceSetDestinationRectangleCommand (pid_t *sender,* int *id,* unsigned int *x,* unsigned int *y,* unsigned int *width,* unsigned int *height*) [inline]

Action Taken This command sets the destination region of a surface within the GENIVI LayerManagement

Expected Frequency Called to rearrange applications output.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*x*	x position of surface on layer
in	*y*	y position of surface on layer
in	*width*	width of surface on layer
in	*height*	height of surface on layer

21.8.2.44 SurfaceSetDimensionCommand::SurfaceSetDimensionCommand (pid_t *sender*, int *id*, unsigned int *width*, unsigned int *height*) [inline]

Action Taken This command sets the dimension of a surface within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*width*	width of surface
in	*height*	height of surface

21.8.2.45 SurfaceSetKeyboardFocusCommand::SurfaceSetKeyboardFocusCommand (pid_t *sender*, unsigned int *surfId*) [inline]

Action Taken This command sets the keyboard focus on a particular surface

Expected Frequency Called whenever a surface needs to receive keyboard events

Parameters

in	*sender*	process id of application that sent this command
in	*surfId*	id of surface

21.8.2.46 SurfaceSetNativeContentCommand::SurfaceSetNativeContentCommand (pid_t *sender*, unsigned int *surfaceId*, unsigned int *handle*, PixelFormat *pixelformat*, uint *OriginalWidth*, uint *OriginalHeight*) [inline]

Action Taken This command sets the native content (application content) of a surface within the GENIVI LayerManagement

Expected Frequency Typically once during startup of an application providing content for a surface.

Parameters

in	*sender*	process id of application that sent this command
in	*surfaceId*	id of surface
in	*handle*	
in	*pixelformat*	
in	*OriginalWidth*	
in	*OriginalHeight*	original height for native content

21.8.2.47 SurfaceSetOpacityCommand::SurfaceSetOpacityCommand (pid_t *sender*, unsigned int *id*, double *Opacity*) [inline]

Action Taken This command sets the opacity of a surface withing the GENIVI LayerManagement

Expected Frequency Can be called very frequently as it can be used for animations.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*Opacity*	opacity of surface

21.8.2.48 SurfaceSetOrientationCommand::SurfaceSetOrientationCommand (pid_t *sender,* unsigned int *id,* OrientationType *Orientation*) [inline]

Action Taken This command sets the orientation of a surface within the GENIVI LayerManagement

Expected Frequency Called to rearrange applications output.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*Orientation*	orientation of surface (rotation)

21.8.2.49 SurfaceSetPositionCommand::SurfaceSetPositionCommand (pid_t *sender,* unsigned int *id,* unsigned int *x,* unsigned int *y*) [inline]

Action Taken This command sets the position of a surface within the GENIVI LayerManagement

Expected Frequency Called for rearranging graphical contents.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*x*	x position of surface on layer
in	*y*	y position of surface on layer

21.8.2.50 SurfaceSetShaderCommand::SurfaceSetShaderCommand (pid_t *sender,* unsigned int *id,* unsigned int *shaderid*) [inline]

Action Taken This command applies a shader to a surface within the GENIVI LayerManagement

Expected Frequency Typically once during surface creation. May be used during runtime for effects.

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*shaderid*	id of shader

21.8.2.51 SurfaceSetSourceRectangleCommand::SurfaceSetSourceRectangleCommand (pid_t *sender,* int *id,* unsigned int *x,* unsigned int *y,* unsigned int *width,* unsigned int *height*) [inline]

Action Taken This command sets the source region of a surface within the GENIVI LayerManagement

Expected Frequency Typically only called at creation of an surface if the graphical output of the application should not be used entirely

Parameters

in	*sender*	process id of application that sent this command
in	*id*	id of surface
in	*x*	x position within surface
in	*y*	y position wihtin surface
in	*width*	width within surface
in	*height*	height within surface

21.8.2.52 SurfaceSetVisibilityCommand::SurfaceSetVisibilityCommand (pid_t *sender,* const unsigned int *givenid,* bool *newvisibility*) [inline]

Action Taken This command sets the visibility of a surface within the GENIVI LayerManagement

Expected Frequency Called more frequently than setOpacity, as event occur which change the general context for the user for example.

Parameters

in	*sender*	process id of application that sent this command
in	*givenid*	id of surface
in	*newvisibility*	TRUE: surface is visible, FALSE: surface is invisible

21.8.2.53 SurfaceUpdateInputEventAcceptance::SurfaceUpdateInputEventAcceptance (pid_t *sender,* unsigned int *surfId,* InputDevice *devices,* bool *accept*) [inline]

Action Taken This command update the list of input devices the surface can accept events from. Call this method if you do not want a surface to receive particular type of event (touch, keyboard, ...)

Expected Frequency Preferably at init. This could lead to weird results if you update the acceptance at runtime, while the surface has already a focus (touch, keyboard, ...)

Parameters

in	*sender*	process id of application that sent this command
in	*surfId*	id of surface
in	*devices*	Bitmask of ilmInputDevice. To set the acceptance status of one or more input device. Note that this method will only the acceptance status for the specified InputDeviced in the "devices" parameter. Not specified input device status will remain unchanged.
in	*accept*	if TRUE, input events from all specified devices will be accepted