CELLULAR SYSTEMS DESIGN SPECIFICATION

Multimedia Framework MSL N5.x Design Specification

Document Revision: 1.0 DRAFT

Issue Date: 12 July 2006

Making**Wireless**

Table of Contents

List of Figures

Revision History

REV	DATE	NOTES
0.1	22 Feb 2006	Initial Draft
0.2	27 Feb 2006	Incorporated the comments after review
0.3	06 March 2006	Updated the API prototypes
1.0	12 July 2006	Updated to the latest implementation

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

1 Products		2 Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

1 Introduction

The Multimedia Service Layer (MSL) is a multimedia application framework layer for camera and imaging features which will enable end users to quickly build applications which can harness camera and imaging capability of Locosto platform. It provides consistent, easy to use use-case-specific API, for use cases like starting a viewfinder with frame overlaying, or zoom or rotate (or combination of all). Every use-case pipeline is created, used and destroyed with similar sets of types and functions to assist the application in using multimedia components.

1.1 Purpose

This purpose of the document is to describe features and APIs for MSL layer implemented on Locosto platform to enable application users to integrate MSL framework with their applications.

1.2 Scope

This document covers features, APIs, integration notes, memory and performance of MSL layer. This document does not cover technical detail on MSL implementation.

1.3 File Name

The file name of this document is cssd_designspec_locosto_mm_msl.doc

1.4 Definitions

MSL	Multimedia Services Layer
OMX	Open Max IL layer
GPF	Generic Protocol Stack Framework
VGA	Video Graphics Array
QCIF	**Q**uarter **C**ommon **I**ntermediate **F**ormat
CIF	**C**ommon **I**ntermediate **F**ormat

2 Overview

MSL layer provides application with three use case pipelines as follows.

1. MSL_UCP_IMGCAP
2. MSL_UCP_IMGTHMB
3. MSL_UCP_IMGVIEW

Each of these pipelines provides similar set of APIs. The feature list for each of the pipelines is detailed below.

2.1 MSL_UCP_IMGCAP Pipeline Features

This pipeline provides user applications with ability to start a view finder, apply rotation, overlay, zoom features and take a snapshot and save it to FFS or RFS file system. The full feature list is listed below

- Enable camera viewfinder mode.
- Apply overlay, rotation and zoom on viewfinder frames in viewfinder mode. These features can be on viewfinder frames in any combination (i.e. only rotation or rotation plus overlay and so on).
  - Supported rotation values are 0, 90, 180 and 270 degrees.
  - Virtually any zoom value is supported. Zoom values are scaled to 1024 to enable smooth zoom. Zoom value of 1024 is considered no zoom, 2048 2x zoom, 4096 4x zoom etc. The user can specify any value >=1024 to simulate smooth zoom effect.
  - Both rotation and zoom values can be dynamically changed while viewfinder is running, but overlay frame can be changed only when MSL_IMGCAP is in deinit state.
  - Three types of overlay supported. 1) Color key based substitution 2) alpha blending and 3) On screen display blending which involved color key substitution and alpha overlay.

    Note that currently Alpha Blending and Alpha Blending Color Key are not supported.
- Ability to support different picture dimensions in viewfinder mode.
  - Tested dimensions are QQVGA and QCIF
- Take a snapshot while in viewfinder mode.
- Apply rotation, overlay and zoom on snapshot image. By default the effects that are on during viewfinder frames (like rotate values or zoom value) will be applicable for the captured image too. User can turn off any of these features before taking snapshot.
  - When Overlay is not enabled, the raw snapshot is taken in yuyv format.
  - When Overlay is enabled, the raw snapshot is captured in RGB16 format
- Ability to take burst mode snapshot. The user can set the burst count and when user make a call to take snapshot the pipeline will take the burst count number of snapshots continuously. Only the last taken snapshot is displayed on screen. Currently the maximum burst count supported is 3. The limitation is only in static memory allocated on system, not in API or implementation.
- Ability to save snapshot images in FFS or RFS. Using the configuration setting user can specify whether the image needs to be saved on RFS (nand) or FFS (nor) file system.

- Ability to take snapshot without starting a viewfinder. There is a direct API provided in this pipeline to take snapshot without calling viewfinder API first.

- Ability to take snapshot in different dimensions. Note that maximum dimension supported by camera is VGA. Apart from camera limitation, the static memory allocated in system is limited to support maximum VGA images.

  - Tested dimension are QQVGA, QCIF, QVGA, CIF and VGA.

2.2 MSL_UCP_IMGTHMB Pipeline Features

This pipeline provides user application with ability to decode a jpeg image, rescale it down to a different dimension and re-encode them to again jpeg format. This feature is especially useful to display thumbnail images as the time taken to decode and display the original images will be too huge to be in the acceptable range. The full features list of this pipeline is listed below

- The source jpeg file could be of any dimension (less than or equal to VGA).

- The source jpeg file could be in RFS or FFS.

- The destination jpeg file (thumbnail jpeg file) could be saved in RFS or FFS independent of source jpeg file.

- Ability to set quality factor for generated thumbnail image.

- Ability to specify user defined dimension for destination jpeg file. (it should be less than VGA)

2.3 MSL_UCP_ IMGVIEW Pipeline Features

This pipeline provides user application with ability to decode a jpeg image, apply post processing features like rotation, rescale, overlay and display them on screen. The full features list of this pipeline is listed below

- The jpeg file could be of any dimension (less than or equal to VGA).

- The jpeg file could be in RFS or FFS.

- Ability to specify rotate, overlay, zoom features independently or in any combinations (like rotate and zoom) for the displayed image.
  - Supported rotation values are 0, 90, 180 and 270 degrees.
  - Virtually any zoom value is supported. Zoom values are scaled to 1024 to enable smooth zoom. Zoom value of 1024 is considered no zoom, 2048 2x zoom, 4096 4x zoom etc. The user can specify any value >=1024 to simulate smooth zoom effect.
  - Both rotation and zoom values can be dynamically changed while viewfinder is running, but overlay frame can be changed only when MSL_IMGCAP is in deinit state.
  - Three types of overlay supported. 1) color key based substitution 2) alpha blending and 3) On screen display blending which involved color key substitution and alpha overlay.

    Note that currently Alpha Blending and Alpha Blending Color Key are not supported.

- Ability to specify the dimension of displayed image and location in screen

3 API And Data Structures

All three MSL pipeline supports similar set of APIs and data structures. The APIs are classified into two types.

1) Synchronous APIs

   These APIs will return to the caller only after completing the required functionality i.e. they are blocking APIs. In MSL design it was taken care that these APIs takes less than 5ms. The APIs for creating an MSL pipeline, deleting an MSL pipeline and doing the configuration setting belong to this category.

2) Asynchronous APIs

   These APIs will trigger an internal task to carry out the actual processing and return to the caller function immediately. Once the actual processing is complete, a callback function (this function pointer is set using a setconfig API) is returned. This API has parameters to specify the usecase pipeline, the status of functionality it was supposed to complete, and API type for which it was called. The APIs for doing MSL pipeline initialization and doing actual data processing (like taking snapshot, or generating thumbnail) belong to this category.

3.1 Common Data Structures

The common data structures are captured in the top level header file msl_api.h. This header file needs to be included by user applications for all MSL usecase pipelines.

3.1.1 MSL_INDEXTYPES

```
#include <msl_api.h>
```

Data Fields

- MSL_CALLBACKSET_CONFIGINDEX,
- MSL_DISPLAY_CONFIGINDEX,
- MSL_CAMERA_CONFIGINDEX,
- MSL_OVERLAY_CONFIGINDEX,
- MSL_BURSTCOUNT_CONFIGINDEX,
- MSL_ENCFILE_CONFIGINDEX,
- MSL_DECFILE_CONFIGINDEX,
- MSL_ZOOM_CONFIGINDEX,
- MSL_RESCALE_CONFIGINDEX,
- MSL_ROTATE_CONFIGINDEX,
- MSL_SEPIAEFFECT_CONFIGINDEX,
- MSL_GRAYEFFECT_CONFIGINDEX,
- MSL_CROPWINDOW_CONFIGINDEX,
- MSL_ENCQUALITY_CONFIGINDEX,
- MSL_IMGINFO_CONFIGINDEX

Detailed Description

This is an enum type structure passed as second parameter of MSL use case pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function). The setconfig API interprets the parameter value (passed as third parameter) depending on the index value.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig

Field Documentation

MSL_CALLBACKSET_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_CALLBACK function pointer. This function is called by MSL pipeline to notify completion of asynchronous APIs and events.

MSL_DISPLAY_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_DISPLAY_CONFIGTYPE structure. This parameter defines the display configuration (displayed width, height, offsets etc) for the image display.

MSL_CAMERA_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_CAM_CONFIGTYPE structure. This structure defines camera specific parameters like camera capture mode (snapshot or viewfinder), image width, height and format.

MSL_OVERLAY_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_IMG_OVERLAYCONFIG structure. This structure defines overlay parameter like overlay type, overlay image width, overlay image height etc for the overlay mage.

MSL_BURSTCOUNT_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to an integer which sets the burst count for camera snapshot capture. This is applicable only to imgcap pipeline and so MSL_ImgCap_SetConfig API only. By default, the burst count is set to 1. When this config is set and burst count is set to greater than 1, camera pipeline (MSL_ImgCap) will take “burst_count” number of snapshots one after the other and saved. All specified post processing (like rotation, zoom, overlay) is performed on each of the snapshots before saving the jpeg image. The last snapshot taken will be previewed on screen and then the snapshot API will return a callback informing application that snapshot is complete.

MSL_ENCFILE_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_FILE_CONFIGTYPE structure. This structure defines encode file details like file name and file format type (rfs, ffs) etc.

MSL_DECFILE_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_FILE_CONFIGTYPE structure. This structure defines decode file details like file name and file format type (rfs, ffs) etc.

MSL_ZOOM_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to an integer specifying the zoom value to be set. The zoom values are scaled to 1024 and so no zoom value is 1024. User can potentially specify zoom value anything equal to or greater than 1024, but quality will degrade beyond 4x zoom (i.e. when value is set to 4096).

MSL_ MSL_RESCALE_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_RESCALE_CONFIGTYPE. This structure defines rescaling parameters for image. This parameters needs to be set only for MSL_Thmb pipeline to specify the thumbnail image dimension.

MSL_ ROTATE_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_IMG_ROTATETYPE enum type. This value specifies the rotation that needs to be performed on this image.

MSL_SEPIAEFFECT_CONFIGINDEX

This index type is currently not supported in any pipelines. This is set for future expansion.

MSL_ GRAYEFFECT_CONFIGINDEX

This index type is currently not supported in any pipelines. This is set for future expansion.

MSL_ CROPWINDOW_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to MSL_IMG_WINDOWTYPE structure type. This structure defines the crop parameters for the image.

MSL_ ENCQUALITY_CONFIGINDEX

When the index type is set to this value, the third parameter to setConfig API should be a pointer to int pointer, which specifies the JPEG encode quality value. This value could be between 1 to 100.

MSL_IMGINFO_CONFIGINDEX

When the index type is set to this value, the third parameter to getConfig API should be a pointer to MSL_IMGINFO_CONFIGTYPE structure. This structure defines the JPEG image property like width, height, format etc. Note that this is passed to getConfig API which returns the value from MSL pipeline.

3.1.2 MSL_FILETYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_FILETYPE_FFS
- MSL_FILETYPE_RFS

Detailed Description

This enum type is member of MSL_FILE_CONFIGTYPE structure, which is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_ENCFILE_CONFIGINDEX OR MSL_DECFILE_CONFIGINDEX. The enum type defines the file type.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig

Field Documentation

MSL_FILETYPE_FFS

When file type is set to this, MSL pipeline will use FFS API for file IO operations (e.g. file read, file write etc).

MSL_FILETYPE_RFS

When file type is set to this, MSL pipeline will use RFS API for file IO operations (e.g. file read, file write etc).

3.1.3 MSL_CALLBACK

```
#include <msl_api.h>
```

Data Fields

- typedef MSL_VOID (* MSL_CALLBACK) (MSL_HANDLE hMSL, MSL_UCPTYPE tUCPType, MSL_CMDTYPE tCMd, MSL_STATUS tStatus)

Detailed Description

This function pointer type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_CALLBACKSET_CONFIGINDEX. It is mandatory for user application to implement this API. The parameters to this API is covered in the field documentation.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig

Field Documentation

MSL_HANDLE

This is the first parameter passed to this function by MSL pipeline. This parameter contains the handle to MSL usecase pipeline instance.

MSL_UCPTYPE

This is the second parameter passed to this function by MSL pipeline. This parameter contains the MSL pipeline type (one of the three pipelines).

MSL_CMDTYPE

This is the third parameter passed to this function by MSL pipeline. This parameter contains the details on for which API the call back was made for.

MSL_STATUS

This is the fourth parameter passed to this function by MSL pipeline. This parameter contains the status of callback function. The status values are given in MSL_<UCP>_STATUS, where UCP is the pipeline type (could be IMGCAP, IMGVIEW or IMGTHMB).

3.1.4 MSL_UCPTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_UCP_IMGCAP
- MSL_UCP_IMGTHMB
- MSL_UCP_IMGVIEW

Detailed Description

This enum type is passed as the second parameter to MSL_CALLBACK function by MSL usecase pipeline. This value could be used by user application to infer the callback source.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

MSL_ UCP_IMGCAP

When value is set to this, it means that the callback is from image capture pipeline.

MSL_ UCP_IMGTHMB

When value is set to this, it means that the callback is from image thumb generation pipeline.

MSL_ UCP_IMGVIEW

When value is set to this, it means that the callback is from image view pipeline.

.

3.1.5 MSL_CMDTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_CMD_VIEWFINDER
- MSL_ CMD_SNAPSHOT
- MSL_ CMD_GENERATE
- MSL_CMD_VIEW
- MSL_CMD_PAUSE
- MSL_CMD_INIT
- MSL_CMD_DEINIT

Detailed Description

This enum type is passed as the third parameter to MSL_CALLBACK function by MSL usecase pipeline. This value could be used by user application to infer the callback scenario.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

MSL_CMD_VIEWFINDER

When third parameter is set to this in the callback API, the fourth parameter will inform the status of viewfinder start. If the viewfinder started successfully, then the fourth parameter would be MSL_IMGCAP_STATUS_OK else it will be set to appropriate error values.

MSL_CMD_SNAPSHOT

When third parameter is set to this in the callback API, the fourth parameter will inform the status of snapshot completion. If the snapshot is successfully completed the fourth parameter would be MSL_IMGCAP_STATUS_OK else it will be set to appropriate error values.

MSL_CMD_GENERATE

When third parameter is set to this in the callback API, the fourth parameter will inform the status of image thumbnail generation. If the thumbnail generation is successful, the fourth parameter would be MSL_IMGTHMB_STATUS_OK else it will be set to appropriate error values.

MSL_CMD_VIEW

When third parameter is set to this in the callback API, the fourth parameter will inform the status of image view completion. If the image was displayed successfully, the fourth parameter of the callback function would be set to MSL_IMGVIEW_STATUS_OK else appropriate error values will be returned.

MSL_CMD_PAUSE

This value is never returned in the current implementation. This is reserved for future implementation.

MSL_CMD_INIT

When third parameter is set to this in the callback API, the fourth parameter will inform the status of pipeline initialization API. If the initialization was successful, the fourth parameter of the callback function would be set to MSL_<UCP>_STATUS_OK else appropriate error values will be returned

MSL_CMD_DEINIT

When third parameter is set to this in the callback API, the fourth parameter will inform the status of pipeline de-initialization API. If the de-initialization was successful, the fourth parameter of the callback function would be set to MSL_<UCP>_STATUS_OK else appropriate error values will be returned

3.1.6 MSL_OVERLAY_MODE

```
#include <msl_api.h>
```

Data Fields

- MSL_OVERLAYMODE_NOOVERLAY
- MSL_OVERLAYMODE_OVERLAP
- MSL_OVERLAYMODE_COLORKEY
- MSL_OVERLAYMODE_ALPHABLENDING
- MSL_OVERLAYMODE_ALPHABLENDINGANDCOLORKEY

Detailed Description

This enum type is member of MSL_IMG_OVERLAYCONFIG structure, which is passed as third parameter of MSL usecase pipeline's setconfig API (for e.g. MSL_ImgCap_SetConfig function) when index type is set as MSL_OVERLAY_CONFIGINDEX. The enum type defines the overlay mode to be used.

Note that currently Alpha Blending and Alpha Blending Color Key are not supported.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig

Field Documentation

MSL_OVERLAYMODE_NOOVERLAY

When overlay mode is set to this, it means that no overlay will be performed.

MSL_OVERLAYMODE_OVERLAP

When overlay mode is set to this, it means that an overlap overlay will be performed. In this mode the overlay image is blindly copied on top of the image in the specified offset location.

MSL_OVERLAYMODE_COLORKEY

When the overlay mode is set to this, it means that color key based substitution will be performed for overlay. In this overlay mode, user specified color key will be used as an index to transparent region in overlay image.

MSL_OVERLAYMODE_ALPHABLENDING

When the overlay mode is set to this, it means that alpha blending method will be used for overlay. In this mode, the overlay image will be alpha blended (using a separate user specified blend value) on top of actual image.

MSL_OVERLAYMODE_ALPHABLENDINGANDCOLORKEY

When the overlay mode is set to this, it means that alpha blending and color key based substitution method will be used for overlay. In this mode, the overlay image will be alpha blended (using a separate user specified blend value) on top of actual image followed by user specified color key based substitution.

3.1.7 MSL_CAMERA_MODE

```
#include <msl_api.h>
```

Data Fields

- MSL_CAMERAMODE_VF
- MSL_CAMERAMODE_SS

Detailed Description

This enum type is member of MSL_CAM_CONFIGTYPE structure, which is passed as third parameter of MSL usecase pipeline's setconfig API (for e.g. MSL_ImgCap_SetConfig function) when index type is set as MSL_CAMERA_CONFIGINDEX. This value specifies the camera mode when starting the imgcap pipeline.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

MSL_CAMERAMODE_VF

The value is set to this to inform MSL imgcap pipeline to start camera in set in viewfinder mode.

MSL_CAMERAMODE_SS

The value is set to this to inform MSL imgcap pipeline to start camera in set in snapshot mode.

3.1.8 MSL_IMG_ROTATETYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_ROTATE_0
- MSL_ROTATE_90
- MSL_ROTATE_180
- MSL_ROTATE_270

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_ROTATE_CONFIGINDEX. This value specifies the rotation that needs to be performed. Note that this rotation value is absolute (not to previous set rotate value).

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

MSL_ ROTATE _0

When the rotate value is set to this, it means that no rotation will be performed on the image.

MSL_ ROTATE _90

When the rotate value is set to this, it means that 90^0 rotation will be performed on the image.

MSL_ ROTATE _180

When the rotate value is set to this, it means that 180^0 rotation will be performed on the image.

MSL_ ROTATE _270

When the rotate value is set to this, it means that 270^0 rotation will be performed on the image.

3.1.9 MSL_COLOR_FORMATTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_COLOR_YUYV
- MSL_COLOR_RGB565

- MSL_COLOR_YUV444
- MSL_COLOR_YUV420
- MSL_COLOR_RGB444
- MSL_COLOR_MONOCHROME

Detailed Description

This enum type is member of MSL_CAM_CONFIGTYPE and MSL_DISPLAY_CONFIGTYPE structure, which is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function). The enum type defines the format of the image data.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

MSL_COLOR_YUYV

This is a 16bits/pixel color format. The data arrangement is this format is $Y_0U_0Y_1V_1$ i.e. interleaved yuv format.

MSL_COLOR_RGB565

This is a 16bits/pixel color format. The data is arranged in little endian mode with 5-bits for R, followed by 6-bits for G and 5-bits for B.

MSL_COLOR_YUV444

This is a 24 bits/pixel color format. The Y, U and V plane are separate each with dimension of image_width x image_height. Note that MSL pipeline provides single pointer for this (no separate Y, U and V pointers).

MSL_COLOR_YUV420

This is a 12 bits/pixel color format. The Y, U and V plane are separate. Y plane has has dimension of image_width x image_height. U and V planes are subsampled by half in both horizontal and vertical directions. Note that MSL pipeline still provides single pointer for this (no separate Y, U and V pointers).

MSL_COLOR_RGB444

This is a 24 bits/pixel color format with 8bits for R, 8 bits for G and 8 bits for B.

MSL_COLOR_MONOCHROME

This is a 8 bits/pixel color format with only one Y plane.

3.1.10 MSL_IMG_WINDOWTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_U16 nImgXOffset
- MSL_U16 nImgYOffset

MSL_U16 nImgCropWidth

- MSL_U16 nImgCropHeight

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_CROPWINDOW_CONFIGINDEX. This structure specifies the crop parameter for the image. Note that when cropping is used, zoom value is ignored.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

nImgXOffset

The X-dimension offset for the crop image region.

nImgYOffset

The Y-dimension offset for the crop image region.

nImgCropWidth

Width of the crop image region starting from nImgXOffset.

nImgCropHeight

Height of the crop image region, starting from nimgYOffset.

3.1.11 MSL_IMG_OVERLAYCONFIG

```
#include <msl_api.h>
```

Data Fields

- MSL_OVERLAY_MODE tOverlayMode
- MSL_U16 nImgWidth
- MSL_U16 nImgHeight
- MSL_U16 nOverlayXOffset
- MSL_U16 nOverlayYOffset
- MSL_U16 nTransparencyColor
- MSL_U16 nAlpha
- MSL_VOID *pOverlayBuff

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_OVERLAY_CONFIGINDEX. This structure specifies the overlay parameter for the image.

Note that currently Alpha Blending and Alpha Blending Color Key are not supported.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

tOverlayMode

See section on MSL_OVERLAY_MODEfor allowed values.

nImgWidth

Width of the overlay image. Note that width of overlay image cannot be greater than the image width.

nImgHeight

Width of the overlay image. Note that height of the overlay image cannot be greater than the image height.

nOverlayXOffset

The x-offset location in the image from where overlay needs to start.

nOverlayYOffset

The y-offset location in the image from where overlay needs to start.

nTransparencyColor

The transparency color when tOverlayMode is set to MSL_OVERLAYMODE_COLORKEY or MSL_OVERLAYMODE_ALPHABLENDINGANDCOLORKEY.

nAlpha

The alpha blend value when tOverlayMode is set to MSL_OVERLAYMODE_ALPHABLENDING or MSL_OVERLAYMODE_ALPHABLENDINGANDCOLORKEY.

pOverlayBuff

Pointer to overlay buffer.

3.1.12 MSL_CAM_CONFIGTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_COLOR_FORMATTYPE tImgFormat
- MSL_U16 unImgWidth
- MSL_U16 unImgHeight
- MSL_CAMERA_MODE tMode

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_CAMERA_CONFIGINDEX. This structure specifies the configuration parameter for the camera.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

tImgFormat

The image format type for camera capture. See MSL_COLOR_FORMATTYPE for details. In the current implementation, when tMode is MSL_CAMERAMODE_VF, the format should always be set to MSL_COLOR_RGB565. When the tMode is MSL_CAMERAMODE_SS, then format could be MSL_COLOR_RGB565 or MSL_COLOR_YUYV. If overlay needs to be performed for snapshot, then format should always be MSL_COLOR_RGB565.

unImgWidth

The width of the image to be captured.

unImgHeight

Height of the camera image to be captured.

tMode

Camera mode. It should be either MSL_CAMERAMODE_VF or MSL_CAMERAMODE_SS.

3.1.13 MSL_DISPLAY_CONFIGTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_COLOR_FORMATTYPE tImgFormat
- MSL_U16 unDisplayImgWidth
- MSL_U16 unDisplayImgHeight
- MSL_U16 unDisplayXOffset
- MSL_U16 unDisplayYOffset

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_DISPLAY_CONFIGINDEX. This structure specifies the configuration parameter for the displayed image on LCD.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

tImgFormat

The image format type for image to be displayed on LCD screen. See MSL_COLOR_FORMATTYPE for details. In the current implementation, the format should always be set to MSL_COLOR_RGB565.

unDisplayImgWidth

The width of the image that needs to be displayed on LCD. This image width should be less than or equal the width of the LCD display. MSL pipeline will do the necessary rescaling when the camera captured image (applicable for IMGCAP pipeline) or decoded image (applicable for IMGVIEW pipeline) height is different from display width.

unDisplayImgHeight

The height of the image that needs to be displayed on LCD. This image height should be less than or equal the height of the LCD display. MSL pipeline will do the necessary rescaling when the camera captured image (applicable for IMGCAP pipeline) or decoded image (applicable for IMGVIEW pipeline) height is different from display height.

unDisplayXOffset

X-offset in screen where the image should be displayed.

unDisplayYOffset

Y-offset in screen where the image should be displayed.

3.1.14 MSL_RESCALE_CONFIGTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_U16 unRescaledImgWidth
- MSL_U16 unRescaledImgHeight

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_RESCALE_CONFIGINDEX. This structure specifies the rescaled image dimension when rescale needs to be performed on the image. This index is supported only in the thumbnail generation pipeline.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

unRescaledImgWidth

Width of the rescaled image.

unRescaledImgHeight

Height of the rescaled image.

3.1.15 MSL_FILE_CONFIGTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_STRING sFileName
- MSL_FILETYPE tFileType

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's setconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_ENCFILE_CONFIGINDEX or MSL_DECFILE_CONFIGINDEX. This structure contains the attributes of the file to be encoded or decoded.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

sFileName

This is a pointer to an array containing the fully qualified name of the file.

tFileType

Specified the file type i.e. RFS or FFS.

3.1.16 MSL_IMGINFO_CONFIGTYPE

```
#include <msl_api.h>
```

Data Fields

- MSL_U16 nExtendedImgWidth;
- MSL_U16 nExtendedImgHeight;
- MSL_U16 nActualImgWidth;
- MSL_U16 nActualImgHeight;
- MSL_COLOR_FORMATTYPE tColorFormat;

Detailed Description

This enum type is passed as third parameter of MSL usecase pipeline's getconfig API (for eg MSL_ImgCap_SetConfig function) when index type is set as MSL_IMGINFO_CONFIGINDEX. This structure contains detail information about the image decoded.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

nExtendedImgWidth

Width of the image including the padded dimension.

nExtendedImgHeight

Height of the image including the padded dimension.

nActualImgWidth

Actual width of the image excluding the padded dimension.

nActualImgHeight

Actual height of the image excluding the padded dimension.

tColorFormat

The color format of the JPEG image

3.1.17 MSL_<UCP>_STATUS

```
#include <msl_api.h>
```

Data Fields

- MSL##_UCP_##STATUS_OK = 0, \
- MSL##_UCP_##STATUS_EOS, \
- MSL##_UCP_##ERROR_UNKNOWN, \
- MSL##_UCP_##ERROR_NOT_IMPLEMENTED, \
- MSL##_UCP_##ERROR_INVALID_STATE, \
- MSL##_UCP_##ERROR_INVALID_ARGUMENT, \
- MSL##_UCP_##ERROR_INVALID_UCP, \
- MSL##_UCP_##ERROR_INVALID_HANDLE, \
- MSL##_UCP_##ERROR_NOMEMORY, \
- MSL##_UCP_##ERROR_BAD_STREAM, \
- MSL##_UCP_##ERROR_IOREAD, \
- MSL##_UCP_##ERROR_IOWRITE, \
- MSL##_UCP_##BASE_LAST_COMMON

Detailed Description

This enum type is the return value for all MSL APIs (both synchronous and asynchronous). This enum type is also returned as the last (fourth) parameter of MSL_CALLBACK function.

##_UCP_## could be one of IMGCAP, IMGVIEW or IMGTHMB depending on the use case.

See also:

MSL_ImgCap_SetConfig, MSL_ImgThmb_SetConfigs, MSL_ImgView_SetConfig, MSL_CALLBACK

Field Documentation

MSL##_UCP_##STATUS_OK

This status means there is no error.

MSL##_UCP_##STATUS_EOS

This status is not currently returned by MSL pipelines.

MSL##_UCP_##ERROR_UNKNOWN

This status is returned when MSL pipeline encounters unknown errors. This is a fatal error.

MSL##_UCP_##ERROR_NOT_IMPLEMENTED

This status is returned when MSL do not support the specific feature.

MSL##_UCP_##ERROR_INVALID_STATE

This status is returned when MSL usecase pipeline is not in a state to support the request.

MSL##_UCP_##ERROR_INVALID_ARGUMENT

This status is returned when MSL usecase pipeline do not recognize the command.

MSL##_UCP_##ERROR_INVALID_UCP

This status is not returned in the current implementation.

MSL##_UCP_##ERROR_INVALID_HANDLE

This status is returned when the handle provided to pipeline is invalid.

MSL##_UCP_##ERROR_NO_MEMORY

This status is returned when there is no memory to create a usecase pipeline.

MSL##_UCP_##ERROR_BAD_STREAM

This status is returned when the MSL pipeline encounters a file open error.

MSL##_UCP_##ERROR_IOREAD

This status is returned when the MSL pipeline encounters a file read error.

MSL##_UCP_##ERROR_IOWRITE

This status is returned when the MSL pipeline encounters a file write error.

3.2 MSL IMGCAP Specific Data Structures

None

3.3 MSL IMGVIEW Specific Data Structures

None

3.4 MSL IMGVIEW Specific Data Structures

None

3.5 COMMON APIs

TBD

3.6 MSL IMGCAP APIs

This pipeline needs to be created for camera preview and image capture applications.

3.6.1 MSL_ImgCap_Create

```
MSL_IMGCAP_STATUS MSL_ImgCap_Create (MSL_HANDLE *phIMGCap);
```

Implementation

This API creates an instance of Image capture pipeline and returns a handle to the instance. This is a synchronous API.

Parameters

phIMGCap	Pointer to a handle to MSL_HANDLE. This pointer will be filled with a valid handle if the call is successful.

Return

MSL_IMGCAP_STATUS	The possible return values are MSL_IMGCAP_STATUS_OK or MSL_IMGCAP_ERROR_NOMEMORY. The former is returned for a successful creation and the later when there is no memory for creating an instance.

Pre Conditions

phIMGCap	Need to ensure that the system has enough memory to carry out the requirement.

Post Conditions

phIMGCap	None.

See Also MSL_ImgCap_Delete

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

The most common reason for returning an error is insufficient memory in the system. The locosto system uses static memory allocation which uses memory pools. In the current implementation, MSL shares the memory pool and hence uses BspGroupHandle as the memory pool handle for allocation.

3.6.2 MSL_ImgCap_SetConfig

```
MSL_IMGCAP_STATUS MSL_ImgCap_SetConfig(MSL_HANDLE hIMGCap, MSL_INDEXTYPES
tIndex, MSL_VOID *pParam;
```

Implementation

This API should be called to set the configuration for image capture/viewfinder scenario. This API needs to be called multiple times to set different parameter. The parameters are identified by tIndex type value passed to this function. The index types are classified into mandatory, optional, dynamically configurable (i.e. it can be called anytime after Create), or non-dynamic configurable (i.e. it can be set only before calling MSL_ImgCap_Init API. This is a synchronous API.

Parameters

hIMGCap	MSL_HANDLE handle returned by MSL_ImgCap_Create.
tIndex	MSL_INDEXTYPES. The set of index values supported is given in the Table 1below.
pParam	This is a pointer to parameter value. The parameter depends on tIndex types.

Return

MSL_IMGCAP_STATUS	The possible return values are MSL_IMGCAP_STATUS_OK, MSL_IMGCAP_ERROR_INVALID_STATE, MSL_IMGCAP_ERROR_NOMEMORY or MSL_IMGCAP_ERROR_INVALID_ARGUMENT. OK status is returned for a successful configuration settings.

Pre Conditions

hIMGCap should be a valid handle.

Post Conditions

None.

See Also

None

Table 1 Supported MSL_INDEXTYPES in image capture pipeline

Index Type	Mandatory	Can be called dynamically
MSL_CALLBACKSET_CONFIGINDEX	Yes	No
MSL_CAMERA_CONFIGINDEX	Yes	No
MSL_DISPLAY_CONFIGINDEX	Yes	No
MSL_OVERLAY_CONFIGINDEX	No	No
MSL_CROPWINDOW_CONFIGINDEX	No	Yes
MSL_ENCFILE_CONFIGINDEX	Yes	Yes
MSL_ENCQUALITY_CONFIGINDEX	No	Yes
MSL_ZOOM_CONFIGINDEX	No	Yes
MSL_ROTATE_CONFIGINDEX	No	Yes
MSL_BURSTCOUNT_CONFIGINDEX	No	Yes

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

To start the pipeline for viewfinder applications, the camera mode set using MSL_CAMERA_CONFIGINDEX should be MSL_CAMERAMODE_VF. To start the pipeline for snapshot applications the camera mode should be set to MSL_CAMERAMODE_SS. The user can directly start in snapshot mode or viewfinder mode, but when switching between the modes, first MSL_ImgCap_Deinit should be called, followed by MSL_ImgCap_SetConfig (to change the camera mode) and then MSL_ImgCap_Init again.

3.6.3 MSL_ImgCap_Init

```
MSL_IMGCAP_STATUS MSL_ImgCap_Init (MSL_HANDLE hIMGCap);
```

Implementation

This function does the initialization of ImgCapture pipeline. This is an asynchronous API.

Parameters

hIMGCap	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGCAP_STATUS	The possible return values are MSL_IMGCAP_STATUS_OK or MSL_IMGCAP_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hIMGCap should be a valid handle and all mandatory configurations should be set using MSL_ImgCap_SetConfig API.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

If the initialization is successful, it will make call MSL_CALLBACK API with following parameters. (_hMSLIMGCAP, MSL_UCP_IMGCAP, MSL_CMD_INIT, MSL_IMGCAP_STATUS_OK). If the initialization is not successful, the first three parameters will still remain same, but the last parameter will contain the appropriate error value.

3.6.4 MSL_ImgCap_ Viewfinder

```
MSL_IMGCAP_STATUS MSL_ImgCap_Viewfinder (MSL_HANDLE hIMGCap);
```

Implementation

This function starts camera in viewfinder mode and starts displaying images in LCD. This is an asynchronous API.

Parameters

hIMGCap	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGCAP_STATUS	The possible return values are MSL_IMGCAP_STATUS_OK or MSL_IMGCAP_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

MSL_ImgCap_Init call is successfully completed i.e. the asynchronous call back function has returned no error. Also the camera mode should be set to MSL_CAMERAMODE_VF

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

This API returns an asynchronous callback with (_hIMGCAP, MSL_UCP_IMGCAP, MSL_CMD_VIEWFINDER, MSL_IMGCAP_STATUS_OK) if viewfinder was started successfully. The fourth parameter will be different if viewfinder could not be started.

The application needs to call MSL_ImgCap_DeInit to stop the viewfinder or to switch to snapshot mode. Note that rotation, zoom etc can be dynamically updated when viewfinder is running.

3.6.5 MSL_ImgCap_ MSL_ImgCap_Snapshot

```
MSL_IMGCAP_STATUS MSL_ImgCap_ MSL_ImgCap_Snapshot (MSL_HANDLE hIMGCap);
```

Implementation

This function starts camera in viewfinder mode and starts displaying images in LCD. This is an asynchronous API.

Parameters

hIMGCap	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGCAP_STATUS	The possible return values are MSL_IMGCAP_STATUS_OK or MSL_IMGCAP_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

MSL_ImgCap_Init call is successfully completed i.e. the asynchronous call back function has returned no error. Also the camera mode should be set to MSL_CAMERAMODE_SS

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

This API returns an asynchronous callback with (_hIMGCAP, MSL_UCP_IMGCAP, MSL_CMD_SNAPSHOT, MSL_IMGCAP_STATUS_OK) if snapshot was completed successfully. The fourth parameter will be different if pipeline encountered error while taking snapshot

The application needs to call MSL_ImgCap_DeInit to stop the viewfinder or to switch to snapshot mode. Note that zoom, rotate, overlay are preserved when switching from viewfinder to snapshot mode and these values will be applied over snapshot image. If a different value of zoom, rotate or overlay needs to be set, then the user should call MSL_ImgCap_SetConfig with appropriate index before calling Init for snapshot.

3.6.6 MSL_ImgCap_DeInit

```
MSL_IMGCAP_STATUS MSL_ImgCap_DeInit (MSL_HANDLE hIMGCap);
```

Implementation

This function does the deinitialization of ImgCapture pipeline. This is an asynchronous API.

Parameters

hIMGCap	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGCAP_STATUS	The possible return values are MSL_IMGCAP_STATUS_OK or MSL_IMGCAP_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hIMGCap should be a valid and all previous call backs from asynchronous APIs are completed. Given the above condition, this API could be called anytime after MSL_ImgCap_Init.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

This is a mandatory API before calling MSL_ImgCap_Destroy API. This API is the only way to stop viewfinder mode. This API should be called to switch between viewfinder and snapshot modes.

3.6.7 MSL_ImgCap_ Destroy

```
MSL_IMGCAP_STATUS MSL_ImgCap_Destroy (MSL_HANDLE hIMGCap);
```

Implementation

This function destroys all memory allocated for ImgCapture pipeline including the handle. This is an synchronous API.

Parameters

hIMGCap	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGCAP_STATUS	The possible return values are MSL_IMGCAP_STATUS_OK or MSL_IMGCAP_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hIMGCap should be a valid and MSL_ImgCap_DeInit should be called prior to this.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

3.7 MSL IMGVIEW APIs

This pipeline needs to be created for image viewing.

3.7.1 MSL_ImgView_Create

```
MSL_IMGVIEW_STATUS MSL_ImgView_Create (MSL_HANDLE * phIMGView);
```

Implementation

This API creates an instance of Image viewer pipeline and returns a handle to the instance. This is a synchronous API.

Parameters

phIMGView	Pointer to a handle to MSL_HANDLE. This pointer will be filled with a valid handle if the call is successful.

Return

MSL_IMGVIEW_STATUS	The possible return values are MSL_ IMGVIEW _STATUS_OK or MSL_ IMGVIEW _ERROR_NOMEMORY. The former is returned for a successful creation and the later when there is no memory for creating an instance.

Pre Conditions

Need to ensure that the system has enough memory to carry out the requirement.

Post Conditions

None.

See Also MSL_ ImgView_Delete

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

The most common reason for returning an error is insufficient memory in the system. The locosto system uses static memory allocation which uses memory pools. In the current implementation, MSL shares the memory pool and hence uses BspGroupHandle as the memory pool handle for allocation.

3.7.2 MSL_ ImgView_SetConfig

```
MSL_IMGVIEW_STATUS MSL_ImgView_SetConfig(MSL_HANDLE ImgView,
MSL_INDEXTYPES tIndex, MSL_VOID *pParam;
```

Implementation

This API should be called to set the configuration for image view scenario. This API needs to be called multiple times to set different parameter. The parameters are identified by tIndex type value passed to this function. The index types are classified into mandatory, optional, dynamically configurable (i.e. it can be called anytime after Create), or non-dynamic configurable (i.e. it can be set only before calling MSL_ImgCap_Init API. This is a synchronous API.

Parameters

hIMGView	MSL_HANDLE handle returned by MSL_ImgView _Create.
tIndex	MSL_INDEXTYPES. The set of index values supported is given in the Table 1below.
pParam	This is a pointer to parameter value. The parameter depends on tIndex types.

Return

MSL_ IMGVIEW_STATUS	The possible return values are MSL_IMGVIEW_STATUS_OK, MSL_IMGVIEW_ERROR_INVALID_STATE, MSL_IMGVIEW_ERROR_NOMEMORY or MSL_IMGVIEW__ERROR_INVALID_ARGUMENT. OK status is returned for a successful configuration settings.

Pre Conditions

hIMGView should be a valid handle.

Post Conditions

None.

See Also

None

Table 2 Supported MSL_INDEXTYPES in image view pipeline

Index Type	Mandatory	Can be called dynamically
MSL_CALLBACKSET_CONFIGINDEX	Yes	No
MSL_DISPLAY_CONFIGINDEX	Yes	Yes
MSL_OVERLAY_CONFIGINDEX	No	Yes
MSL_CROPWINDOW_CONFIGINDEX	No	Yes
MSL_ZOOM_CONFIGINDEX	No	Yes
MSL_ROTATE_CONFIGINDEX	No	Yes
MSL_DECFILE_CONFIGINDEX	Yes	Yes

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

3.7.3 MSL_ ImgView_Init

```
MSL_IMGVIEW_STATUS MSL_ImgView_Init (MSL_HANDLE hIMGView);
```

Implementation

This function does the initialization of image view pipeline. This is an asynchronous API.

Parameters

hIMGView	MSL_HANDLE handle returned by MSL_ImgView_Create..

Return

MSL_ IMGVIEW_STATUS	The possible return values are MSL_ IMGVIEW_STATUS_OK or MSL_IMGVIEW_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hIMGView should be a valid handle and all mandatory configurations should be set using MSL_ImgView_SetConfig API.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

If the initialization is successful, it will make call MSL_CALLBACK API with following parameters. (_hMSLIMGCAP, MSL_UCP_IMGVIEW, MSL_CMD_INIT, MSL_IMGVIEW_STATUS_OK). If the initialization is not successful, the first three parameters will still remain same, but the last parameter will contain the appropriate error value.

3.7.4 MSL_ImgView_View

```
MSL_IMGVIEW_STATUS MSL_ImgView_View (MSL_HANDLE hIMGView);
```

Implementation

This function decodes the jpeg image, does rotation, overlay, zoom, crop as set by s MSL_ImgView Setconfig API and displays images on LCD. This is an asynchronous API.

Parameters

hIMGView	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGVIEW_STATUS	The possible return values are MSL_IMGVIEW_STATUS_OK or MSL_ IMGVIEW _ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

MSL_ImgCap_Init call is successfully completed i.e. the asynchronous call back function has returned no error.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

This API returns an asynchronous callback with (_hIMGView, MSL_UCP_IMGVIEW, MSL_CMD_VIEW, MSL_IMGVIEW_STATUS_OK) after successfully displaying the image.

3.7.5 MSL_ImgView_Deinit

```
MSL_IMGVIEW_STATUS MSL_ImgCap_DeInit (MSL_HANDLE hIMGView);
```

Implementation

This function does the deinitialization of image viewer pipeline. This is an asynchronous API.

Parameters

hIMGView	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGVIEW_STATUS	The possible return values are MSL_IMGVIEW_STATUS _OK or MSL_IMGVIEW_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hIMGView should be a valid and all previous call backs from asynchronous APIs are completed. Given the above condition, this API could be called anytime after MSL_ImgView_Init.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

This is a mandatory API before calling MSL_ImgView_Destroy API.

3.7.6 MSL_ImgView_ Destroy

```
MSL_IMGCAP_STATUS MSL_ImgView_Destroy (MSL_HANDLE hIMGView);
```

Implementation

This function destroys all memory allocated for image view pipeline including the handle. This is an synchronous API.

Parameters

hIMGView	MSL_HANDLE handle returned by MSL_ImgCap_Create..

Return

MSL_IMGVIEW_STATUS	The possible return values are MSL_IMGVIEW_STATUS_OK or MSL_IMGVIEW_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hIMGView should be a valid and MSL_ImgView_DeInit should be called prior to this.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

3.8 MSL IMGTHMB APIs

This pipeline needs to be creating thumbnail jpeg images. Thumbnail jpeg images are normal jpeg files, but of smaller dimensions so that they can be displayed quicker in an thumbnail viewer application.

3.8.1 MSL_ImgThmb_Create

```
MSL_IMGTHMB_STATUS MSL_ImgThmb_Create (MSL_HANDLE * phMSLIMGThmb);
```

Implementation

This API creates an instance of Image viewer pipeline and returns a handle to the instance. This is a synchronous API.

Parameters

phMSLIMGThmb	Pointer to a handle to MSL_HANDLE. This pointer will be filled with a valid handle if the call is successful.

Return

MSL_IMGVIEW_STATUS	The possible return values are MSL_IMGTHMB_STATUS_OK or MSL_IMGTHMB_ERROR_NOMEMORY. The former is returned for a successful creation and the later when there is no memory for creating an instance.

Pre Conditions

Need to ensure that the system has enough memory to carry out the requirement.

Post Conditions

None.

See Also MSL_ ImgThmb_Delete

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

The most common reason for returning an error is insufficient memory in the system. The locosto system uses static memory allocation which uses memory pools. In the current implementation, MSL shares the memory pool and hence uses BspGroupHandle as the memory pool handle for allocation.

3.8.2 MSL_ ImgThmb_SetConfig

```
MSL_IMGTHMB_STATUS MSL_ImgThmb_SetConfig(MSL_HANDLE ImgView,
MSL_INDEXTYPES tIndex, MSL_VOID *pParam;
```

Implementation

This API should be called to set the configuration for image view scenario. This API needs to be called multiple times to set different parameter. The parameters are identified by tIndex type value passed to this function. The index types are classified into mandatory, optional, dynamically configurable (i.e. it can be called anytime after Create), or non-dynamic configurable (i.e. it can be set only before calling MSL_ImgThmb_Init API. This is a synchronous API.

Parameters

hMSLIMGThmb	MSL_HANDLE handle returned by MSL_ImgThmb _Create.
tIndex	MSL_INDEXTYPES. The set of index values supported is given in the Table 3 below.
pParam	This is a pointer to parameter value. The parameter depends on tIndex types.

Return

MSL_ IMGTHMB_STATUS	The possible return values are MSL_IMGTHMB_STATUS_OK, MSL_IMGTHMB_ERROR_INVALID_STATE, MSL_IMGTHMB_ERROR_NOMEMORY or MSL_IMGTHMB__ERROR_INVALID_ARGUMENT. OK status is returned for a successful configuration settings.

Pre Conditions

hMSLIMGThmb should be a valid handle.

Post Conditions

None.

See Also

None

Table 3 Supported MSL_INDEXTYPES in image thumb pipeline

Index Type	Mandatory	Can be called dynamically
MSL_CALLBACKSET_CONFIGINDEX	Yes	No
MSL_DECFILE_CONFIGINDEX	Yes	Yes
MSL_ENCFILE_CONFIGINDEX	Yes	Yes

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

3.8.3 MSL_ ImgThmb_Init

```
MSL_IMGVIEW_STATUS MSL_ImgView_Init (MSL_HANDLE hIMGView);
```

Implementation

This function does the initialization of image thumb pipeline. This is an asynchronous API.

Parameters

hMSLIMGThmb	MSL_HANDLE handle returned by MSL_ImgThmb_Create..

Return

MSL_ IMGVIEW_STATUS	The possible return values are MSL_ IMGTHMB_STATUS_OK or MSL_IMGTHMB_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

HMSLIMGThmb should be a valid handle and all mandatory configurations should be set using MSL_ImgThmb_SetConfig API.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

If the initialization is successful, it will make call MSL_CALLBACK API with following parameters. (_hMSLIMGCAP, MSL_UCP_IMGTHMB, MSL_CMD_INIT, MSL_IMGTHMB_STATUS_OK). If the initialization is not successful, the first three parameters will still remain same, but the last parameter will contain the appropriate error value.

3.8.4 MSL_ImgThmb_Generate

```
MSL_IMGVIEW_STATUS MSL_ImgThmb_Generate (MSL_HANDLE hMSLIMGThmb);
```

Implementation

This function decodes the jpeg image downscales it by user specified value and re-encodes it again.

Parameters

hMSLIMGThmb	MSL_HANDLE handle returned by MSL_ImgThmb_Create..

Return

MSL_IMGTHMB_STATUS	The possible return values are MSL_IMGVIEW_STATUS_OK or MSL_ IMGVIEW _ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

MSL_ImgThmb_Init call is successfully completed i.e. the asynchronous call back function has returned no error.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

This API returns an asynchronous callback with (_hMSLIMGThmb, MSL_UCP_IMGTHMB, MSL_CMD_GENERATE, MSL_IMGTHMB_STATUS_OK) after successfully encoding the thumbnail image.

3.8.5 MSL_ImgThmb_Deinit

```
MSL_IMGTHMB_STATUS MSL_ImgThmb_DeInit (MSL_HANDLE hMSLIMGThmb);
```

Implementation

This function does the deinitialization of image thumb pipeline. This is an asynchronous API.

Parameters

hMSLIMGThmb	MSL_HANDLE handle returned by MSL_ImgThmb_Create..

Return

MSL_IMGVIEW_STATUS	The possible return values are MSL_IMGTHMB_STATUS _OK or MSL_IMGTHMB_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hMSLIMGThmb should be a valid and all previous call backs from asynchronous APIs are completed. Given the above condition, this API could be called anytime after MSL_ImgVThmb_Init.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

This is a mandatory API before calling MSL_ImgThmb_Destroy API.

3.8.6 MSL_ImgThmb_ Destroy

```
MSL_IMGCAP_STATUS MSL_ImgThmb_Destroy (MSL_HANDLE hMSLIMGThmb);
```

Implementation

This function destroys all memory allocated for image thumb pipeline including the handle. This is an synchronous API.

Parameters

hIMGView	MSL_HANDLE handle returned by MSL_ImgThmb_Create..

Return

MSL_IMGTHMB_STATUS	The possible return values are MSL_IMGTHMB_STATUS_OK or MSL_IMGTHMB_ERROR_INVALID_STATE. The former is returned for a successful creation and the later when the call sequence is not proper.

Preconditions

hMSLIMGThmb should be a valid and MSL_ImgVThmb_DeInit should be called prior to this.

Post conditions

None.

Requirement Coverage

This method addresses requirement(s): [SR number(s)]

Notes

4 Call Sequences

TBD

This section is not complete.

Steps for thumbnail creation:

1. Read from NAND the JPEG file in chunks.
2. JPEG decoding of the stream in stripe mode
3. Downscale to QCIF
4. JPEG encoding on the QCIF buffer
5. Save on NAND
6. Once the files ".thu" are created, they can be used to generate the picture gallery image.

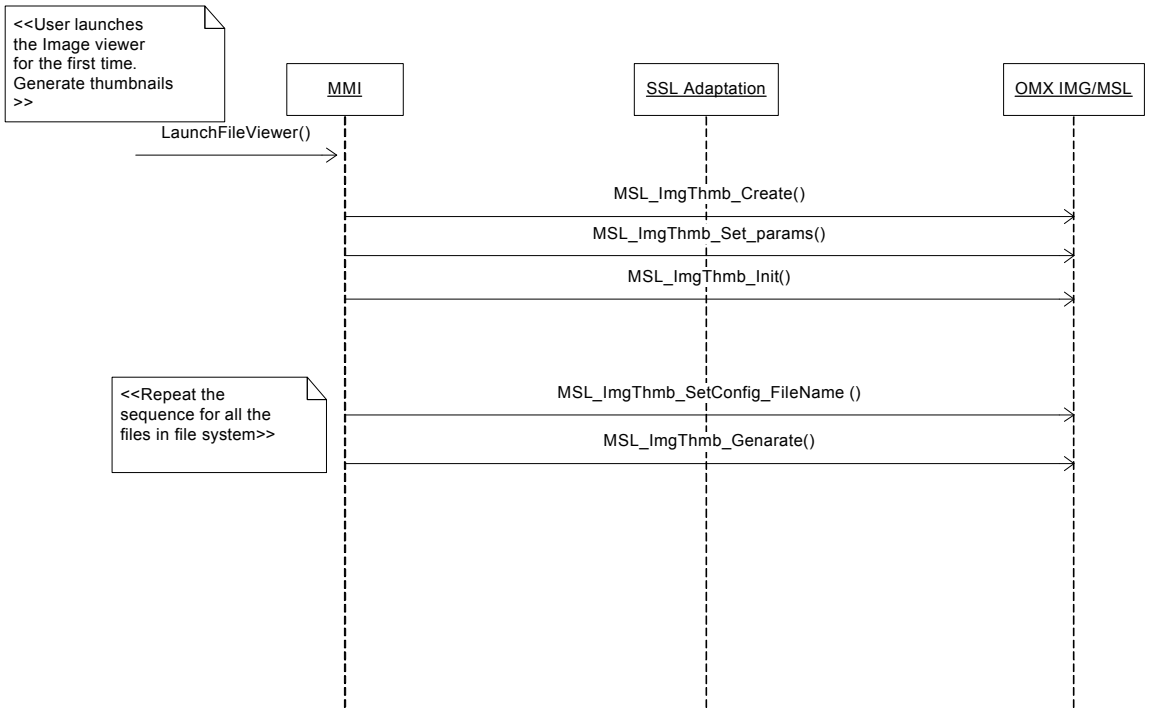

Figure 1. Thumbnail Creation sequence diagram

Image Viewer:

1. Steps for previewing the thumbnail file to full screen:
2. Read the thumbnail file from NAND
3. JPEG decoding on the QCIF
4. Color Convert from YUV to RGB565
5. Display

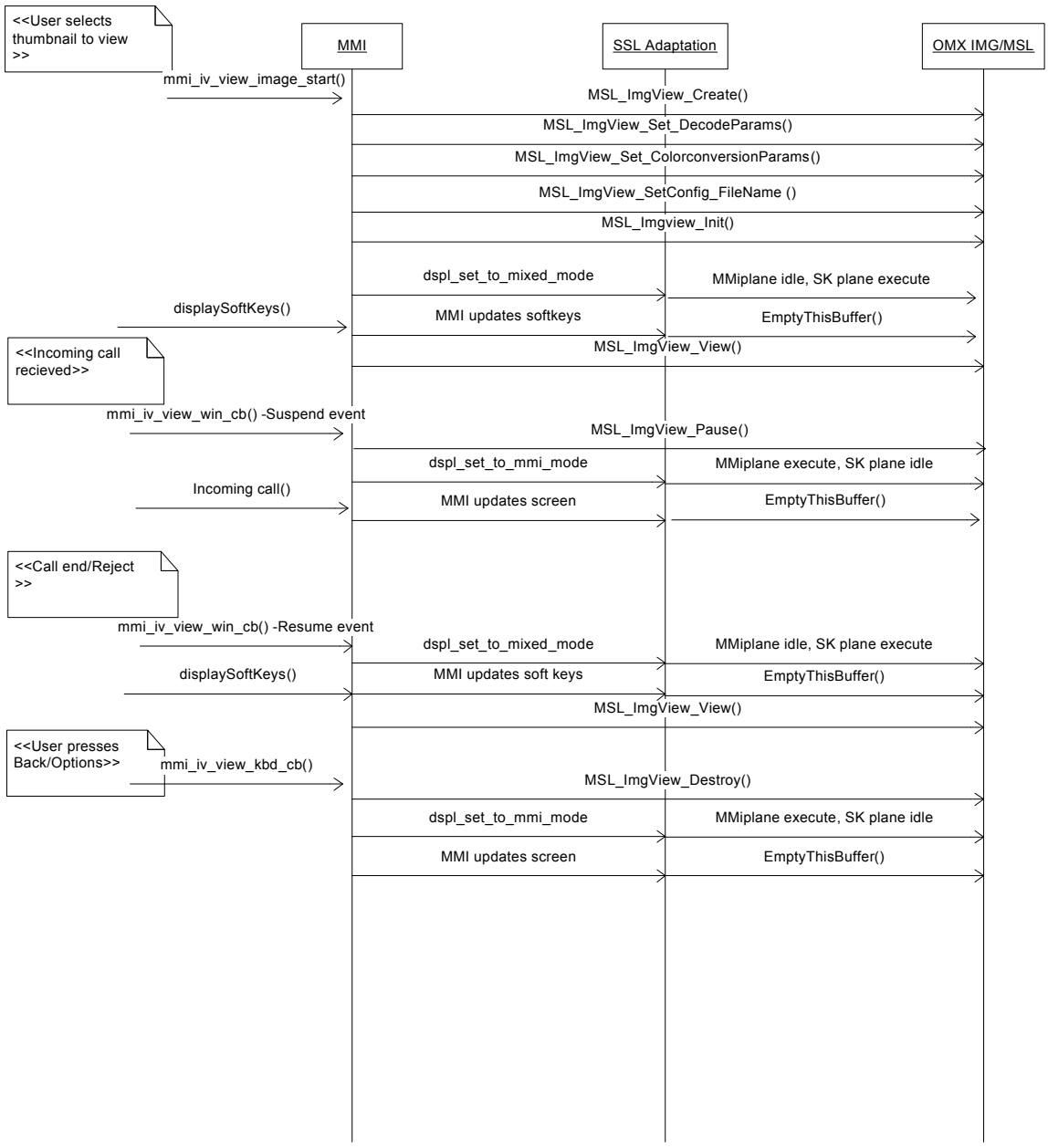

Figure 2. Image Viewer sequence diagram

5 Performance Data

Detail performance data is captured in the *cssd_performancedata_locosto_mm_msl.xls* document which is released along with the software.

6 Memory Requirements

- Total ROM for codec - 24.2 Kbytes
- Total Flash size for codec - 24 Kbytes (it is not 5K)
- Total internal ram size for codec - 5.5 Kbytes (some portion of the code runs from internal memory)
- Total Flash code size for MM framework - 30.3 Kbytes

Apart from this the whole MM framework needs (614K x 2 + 150K x 3 = 1.52MBytes) of RAM space for processing the VGA image data.