DSP/BIOS™ LINK

LNK 018 REF

Version 1.10

MAR 03, 2004

This page has been intentionally left blank.

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.

Mailing Address:

Texas Instruments

Post Office Box 655303

Dallas, Texas 75265

This page has been intentionally left blank.

TABLE OF CONTENTS

INTRODUCTION

1 Purpose & Scope

DSP/BIOS™ LINK is foundation software for the inter-processor communication across the GPP-DSP boundary. It provides a generic API that abstracts the characteristics of the physical link connecting GPP and DSP from the applications. It eliminates the need for customers to develop such link from scratch and allows them to focus more on application development.

2 Text Conventions

<style/bullet>	<definition or explanation>
O	This bullet indicates important information. Please read such text carefully.
q	This bullet indicates additional information.

DSP/BIOS LINK(TM) API

3 dsplink.h

Defines data types and structures used by DSP/BIOS(tm) Link.

Path

$(DSPLINK)\gpp\inc\Linux

Revision

00.10

CONSTANTS

3.1 MAX_PROCESSORS

Maximum number of DSPs supported by DSP/BIOS Link.

Syntax

```
#define MAX_PROCESSORS   1
```

See Also

3.2 MAX_CHANNELS

Maximum number of channels per link supported by DSP/BIOS Link.

Syntax

```
#define MAX_CHANNELS    16
```

See Also

3.3 MAX_ALLOC_BUFFERS

Maximum number of buffers that can be allocated through CHNL_AllocateBuffer.

Syntax

```
#define MAX_ALLOC_BUFFERS      1000
```

See Also

3.4 WAIT_FOREVER

Wait indefinitely.

Syntax

```
#define WAIT_FOREVER           (~((Uint32) 0))
```

See Also

3.5 WAIT_NONE

Do not wait.

Syntax

```
#define WAIT_NONE              ((Uint32) 0)
```

See Also

3.6 NUM_MSGQ_CHANNELS

Number of channels being used for messaging.

Syntax

```
if defined (MSGQ_COMPONENT)
#define NUM_MSGQ_CHANNELS    2
#else /* if defined (MSGQ_COMPONENT) */
#define NUM_MSGQ_CHANNELS    0
#endif /* if defined (MSGQ_COMPONENT) */
```

See Also

3.7 DSP_MAX_STRLEN

Maximum length of string.

Syntax

```
#define DSP_MAX_STRLEN      32
```

See Also

dsplink.h

ENUMERATIONS

3.8 Endianism

Enumeration of data endianism.

Syntax

```
typedef enum {
    Endianism_Default = 1,
    Endianism_Big     = 2,
    Endianism_Little  = 3
} Endianism ;
```

Enum Values

Endianism_Default	Default endianism - no conversion required.
Endianism_Big	Big endian.
Endianism_Little	Little endian.

See Also

3.9 ProcState

Enumerations to indicate processor states.

Syntax

```
typedef enum {
    ProcState_Unknown = 0,
    ProcState_Loaded  = 1,
    ProcState_Started = 2,
    ProcState_Stopped = 3,
    ProcState_Idle    = 4,
    ProcState_Reset   = 5
} ProcState ;
```

Enum Values

ProcState_Unknown	Unknown (possibly error) processor state.
ProcState_Loaded	Indicates the DSP is loaded.
ProcState_Started	Indicates the DSP is started.
ProcState_Stopped	Indicates the DSP is stopped.
ProcState_Idle	Indicates the DSP is idle.
ProcState_Reset	Indicates the DSP is reset.

See Also

3.10 ChannelMode

Mode of a channel.

Syntax

```
typedef enum {
    ChannelMode_Input  = 0x1,
    ChannelMode_Output = 0x2
} ChannelMode ;
```

Enum Values

ChannelMode_Input	Indicates the channel as an Input channel (from DSP to GPP).
ChannelMode_Output	Indicates the channel as an Output channel (from GPP to DSP).

See Also

3.11 ChannelDataSize

Width of data being sent on channel.

Syntax

```
typedef enum {
    ChannelDataSize_16bits = 1,
    ChannelDataSize_32bits = 2
} ChannelDataSize ;
```

Enum Values

ChannelDataSize_16bits	Indicates the data to be transferred through the channel as 16 bit data.
ChannelDataSize_32bits	Indicates the data to be transferred through the channel as 32 bit data.

See Also

3.12 DspArch

Enumerates the various architectures of DSP supported by DSP/BIOS LINK.

Syntax

```
typedef enum {
    DspArch_Unknown  = 0,
    DspArch_C55x     = 1,
    DspArch_C64x     = 2
} DspArch ;
```

Enum Values

DspArch_Unknown	It indicates that the architecture is not supported.
DspArch_C55x	It indicates that the architecture is C55x.
DspArch_C64x	It indicates that the architecture is C64x.

See Also

TYPE DEFINITIONS & STRUCTURES

3.13 ChannelAttrs

Channel Attributes.

Syntax

```
typedef struct ChannelAttrs_tag {
    Endianism       endianism ;
    ChannelMode     mode      ;
    ChannelDataSize size      ;
} ChannelAttrs ;
```

Fields

Endianism	endianism Endiannism information currently not used.
ChannelMode	mode Mode of channel (Input or output).
ChannelDataSize	size Size of data sent on channel (16 bits or 32 bits).

See Also

3.14 ChannelIOInfo

Information for adding or reclaiming a IO request.

Syntax

```
typedef struct ChannelIOInfo_tag {
    Char8 *   buffer ;
    Uint32    size ;
    Uint32    arg ;
} ChannelIOInfo ;
```

Fields

Char8 *	buffer Buffer pointer.
Uint32	size Size of buffer.
Uint32	arg Argument to receive or send.

See Also

3.15 LoaderObject

This object is used to pass arguments to Loader component.

Syntax

```
typedef struct LoaderObject_tag {
    Pstr            baseImage ;
    DspArch         dspArch   ;
    Endianism       endian    ;
    Uint32          wordSize  ;
} LoaderObject ;
```

Fields

Pstr	baseImage DSP executable file name.
DspArch	dspArch Architecture of the dsp.
Endianism	endian Endianism of the processor.
Uint32	wordSize Word size on the dsp.

See Also

None

3.16 ProcAttr

A placeholder for processor information.

Syntax

```
typedef struct ProcAttr_tag {
    Uint32   timeout ;
} ProcAttr ;
```

Fields

Uint32	timeout Time out associated with a DSP.

See Also

4 proc.h

Defines the interfaces and data structures for the API sub-component PROC.

Path

`$(DSPLINK)\gpp\src\api`

Revision

`00.05`

FUNCTIONS

4.1 PROC_Setup

Sets up the necessary data structures for the PROC sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Setup () ;
```

Arguments

`None`

Return Values

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to memory error.
`DSP_EFAIL`	General failure.

Pre Conditions

None

Post Conditions

None

See Also

`PROC_Destroy`

4.2 PROC_Destroy

Destroys the data structures for the PROC component, allocated earlier by a call to PROC_Setup ().

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Destroy () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

None

Post Conditions

None

See Also

PROC_Setup

4.3 PROC_Attach

Attaches the client to the specified DSP and also initializes the DSP (if required).

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Attach (IN  ProcessorId procId,
             OPT ProcAttr *  attr) ;
```

Arguments

IN	ProcessorId	procId
	DSP identifier.	
OPT	ProcAttr *	attr
	Attributes for the processor on which attach is to be done.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_SALREADYATTACHED	Successful attach. Also, indicates that another client has already attached to DSP.
DSP_EINVALIDARG	Parameter ProcId is invalid.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EFAIL	General failure, unable to attach to processor.
DSP_EWRONGSTATE	Incorrect state for completing the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PROC_Detach

4.4 PROC_Detach

Detaches the client from specified processor.

If the caller is the owner of the processor, this function releases all the resources that this component uses and puts the DSP in an unusable state (from application perspective).

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Detach (IN  ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	DSP identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid ProcId parameter.
DSP_EFAIL	General failure, unable to detach.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EATTACHED	Not attached to the target processor.
DSP_EWRONGSTATE	Incorrect state for completing the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PROC_Attach

4.5 PROC_GetState

Gets the current status of DSP by querying the Link Driver.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_GetState (IN   ProcessorId   procId,
               OUT  ProcState *   procState) ;
```

Arguments

IN	ProcessorId	procId
	DSP identifier.	
OUT	ProcState *	procState
	Placeholder for processor state.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid ProcId and/or procState argument.

Pre Conditions

procId must be valid.

procState must be a valid pointer.

Post Conditions

None

See Also

PROC_Load
PROC_Start
PROC_Stop
PROC_Idle

4.6 PROC_Load

Loads the specified DSP executable on the target DSP.

It ensures that the caller owns the DSP.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Load (IN   ProcessorId  procId,
           IN   Char8 *      imagePath,
           IN   Uint32       argc,
           IN   Char8 **     argv) ;
```

Arguments

IN	ProcessorId	procId
	DSP identifier.	
IN	Char8 *	imagePath
	Full path to the image file to load on DSP.	
IN	Uint32	argc
	Number of argument to be passed to the base image upon start.	
IN	Char8 **	argv
	Arguments to be passed to DSP main application.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid ProcId argument.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EFILE	Invalid base image.
DSP_EFAIL	General failure, unable to load image on DSP.
DSP_EWRONGSTATE	Incorrect state for completing the requested operation.

Pre Conditions

procId must be valid.

imagePath must be a valid pointer.

If argc is 0 then argv must be NULL pointer.

If argc is non-zero then argv must be a valid pointer.

Post Conditions

None

See Also

PROC_Attach
PROC_LoadSection

4.7 PROC_LoadSection

Loads the specified section of DSP executable onto the target DSP.

It ensures that the client owns the DSP.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_LoadSection (IN    ProcessorId  procId,
                  IN    Char8 *      imagePath,
                  IN    Uint32       sectID) ;
```

Arguments

IN	ProcessorId	procId
	DSP identifier.	
IN	Char8 *	imagePath
	Full path to the image file.	
IN	Uint32	sectID
	Section ID of section to load.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid ProcId argument.
DSP_EFILE	Invalid ImagePath parameter.
DSP_EINVALIDSECTION	Invalid section name.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EFAIL	General failure, unable to load section on DSP.
DSP_EWRONGSTATE	Incorrect state for completing the requested operation.

Pre Conditions

procId must be valid.

imagePath must be a valid pointer.

Post Conditions

None

See Also

PROC_Attach
PROC_Load

4.8 PROC_Start

Starts execution of the loaded code on DSP from the starting point specified in the DSP executable loaded earlier by call to PROC_Load ().

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Start (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	DSP Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid ProcId argument.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EFAIL	General failure, unable to start DSP.
DSP_EWRONGSTATE	Incorrect state for completing the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PROC_Attach
PROC_Load
PROC_Stop

4.9 PROC_Stop

Stops the DSP.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Stop (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	DSP Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid ProcId argument.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EFAIL	General failure, unable to stop DSP.
DSP_EWRONGSTATE	Incorrect state for completing the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PROC_Attach
PROC_Load
PROC_Start

4.10 PROC_Control

Provides a hook to perform device dependent control operations on the DSP.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Control (IN  ProcessorId procId,
              IN  Int32       cmd,
              OPT Pvoid       arg) ;
```

Arguments

IN	ProcessorId	procId
	DSP Identifier.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PROC_Attach

4.11 PROC_Instrument

Gets the instrumentation data associated with PROC sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
PROC_Instrument (IN ProcessorId procId, OUT ProcInstrument * retVal) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor for which instrumentation information is to be obtained.	
OUT	ProcInstrument *	retVal
	OUT argument to contain the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	retVal is invalid.

Pre Conditions

procId must be valid.

retVal must be a valid pointer.

Post Conditions

None

See Also

None

4.12 PROC_Debug

Prints the debug information summarizing the current status of the PROC component.

Syntax

```
EXPORT_API
Void
PROC_Debug (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor.	

Return Values

None

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

None

5 chnl.h

Defines the interfaces and data structures for the API sub-component CHNL.

Path

$(DSPLINK)\gpp\src\api

Revision

00.08

FUNCTIONS

5.1 CHNL_Create

Creates resources used for transferring data between GPP and DSP.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_Create (IN ProcessorId      procId,
             IN ChannelId        chnlId,
             IN ChannelAttrs *   attrs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Id to open.	
IN	ChannelAttrs *	attrs
	Channel attributes - if NULL, default attributes are applied.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

Attributes must be valid.

Post Conditions

None

See Also

None

5.2 CHNL_Delete

Releases channel resources used for transferring data between GPP and DSP.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_Delete (IN ProcessorId    procId,
             IN ChannelId      chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

Channels for specified processors should have been initialized.

Processor and channel ids should be valid.

Post Conditions

None

See Also

CHNL_Create

5.3 CHNL_AllocateBuffer

Allocates an array of buffers of specified size and returns them to the client.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_AllocateBuffer (IN  ProcessorId procId,
                     IN  ChannelId   chnlId,
                     OUT Char8 **    bufArray,
                     IN  Uint32      size,
                     IN  Uint32      numBufs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
OUT	Char8 **	bufArray
	Pointer to receive array of allocated buffers.	
IN	Uint32	size
	Size of each buffer.	
IN	Uint32	numBufs
	Number of buffers to allocate.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid parameter passed.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

CHNL_Create
CHNL_FreeBuffer

5.4 CHNL_FreeBuffer

Frees buffer(s) allocated by CHNL_AllocateBuffer.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_FreeBuffer (IN ProcessorId procId,
                 IN ChannelId   chnlId,
                 IN Char8 **    bufArray,
                 IN Uint32      numBufs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Char8 **	bufArray
	Pointer to the array of buffers to be freed.	
IN	Uint32	numBufs
	Number of buffers to be freed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid parameter passed.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

```
CHNL_Create
CHNL_AllocateBuffer
```

5.5 CHNL_Issue

Issues an input or output request on a specified channel.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_Issue (IN ProcessorId      procId,
            IN ChannelId        chnlId,
            IN ChannelIOInfo *  ioReq) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	ChannelIOInfo *	ioReq
	Information regarding IO.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid parameter passed.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

CHNL_Reclaim

5.6 CHNL_Reclaim

Gets the buffer back that has been issued to this channel.

This call blocks for specified timeout value ranging from NO_WAIT to WAIT_FOREVER.

Syntax

```
EXPORT_API
EXPORT_API
DSP_STATUS
CHNL_Reclaim (IN     ProcessorId       procId,
              IN     ChannelId         chnlId,
              IN     Uint32            timeout,
              IN OUT ChannelIOInfo *   ioReq) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Uint32	timeout
	Timeout value for this operation. Unit of timeout is OS dependent.	
IN OUT	ChannelIOInfo *	ioReq
	Information needed for doing reclaim.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid parameter passed.
CHNL_E_NOIOC	Timeout parameter was "NO_WAIT", yet no I/O completions were queued.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

CHNL_Issue

5.7 CHNL_Idle

In the input mode channel, this function resets the channel and causes any currently buffered input data to be discarded.

In the output mode channel, this function causes any currently queued buffers to be transferred through the channel. It causes the client to wait for as long as it takes for the data to be transferred through the channel.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_Idle (IN ProcessorId  procId,
           IN ChannelId    chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid parameter passed.

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

CHNL_Create

5.8 CHNL_Flush

Discards all the requested buffers that are pending for transfer both in case of input mode channel as well as output mode channel.

One must still have to call the CHNL_Reclaim to get back the discarded buffers.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_Flush (IN ProcessorId         procId,
            IN ChannelId           chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid parameter passed.

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

CHNL_Create
CHNL_Issue

5.9 CHNL_Control

Provides a hook to perform device dependent control operations on channels.

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_Control (IN  ProcessorId    procId,
              IN  ChannelId      chnlId,
              IN  Int32          cmd,
              OPT Pvoid          arg) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EINVALIDARG	Invalid parameter passed.

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

None

5.10 CHNL_Instrument

Gets the instrumentation information related to CHNL's

Syntax

```
EXPORT_API
DSP_STATUS
CHNL_Instrument (IN  ProcessorId      procId,
                 IN  ChannelId        chnlId,
                 OUT ChnlInstrument * retVal) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor.	
IN	ChannelId	chnlId
	Identifier for channel for which instrumentation information is to be obtained.	
OUT	ChnlInstrument *	retVal
	OUT argument to contain the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	retVal is invalid.

Pre Conditions

retVal must be a valid pointer.

Post Conditions

None

See Also

None

5.11 CHNL_Debug

Prints the current status of CHNL subcomponent.

Syntax

```
EXPORT_API
Void
CHNL_Debug (IN ProcessorId procId,
            IN ChannelId   chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor.	
IN	ChannelId	chnlId
	Identifier for channel.	

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

6 msgq.h

Defines the interfaces and data structures for the API sub-component MSGQ.

Path

$(DSPLINK)\gpp\src\api

Revision

00.08

FUNCTIONS

6.1 MSGQ_AllocatorOpen

This function initializes an allocator. The allocator id specified in the call should be configured in the CFG. This function should be called only once as part of system initialization after the client has called PROC_Attach the first time. Subsequent tasks / threads should not call this function again as it would result in unpredictable behavior.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_AllocatorOpen (IN  AllocatorId  mqaId, IN  Pvoid mqaAttrs) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be opened.	
IN	Pvoid	mqaAttrs
	Attributes for initialization of the MQA component. The structure of the expected attributes is specific to an MQA.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.
DSP_EACCESSDENIED	Allocator already open.

Pre Conditions

mqaAttrs must be valid.

Post Conditions

None

See Also

MSGQ_AllocatorClose

6.2 MSGQ_AllocatorClose

This function finalizes the allocator component.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_AllocatorClose (IN  AllocatorId mqaId) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be closed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.
DSP_EACCESSDENIED	Not the owner.

Pre Conditions

None

Post Conditions

None

See Also

MSGQ_AllocatorOpen

6.3 MSGQ_TransportOpen

This function initializes an MQT component. The transport id specified in the call should be configured in the CFG. This function should be called only once as part of system initialization after the client has called PROC_Attach the first time. Subsequent tasks / threads should not call this function again as it would result in unpredictable behavior.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_TransportOpen (IN  TransportId mqtId, IN  Pvoid mqtAttrs) ;
```

Arguments

IN	TransportId	mqtId
	ID of the MQT to be opened.	
IN	Pvoid	mqtAttrs
	Attributes for initialization of the MQT component. The structure of the expected attributes is specific to an MQT.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.
DSP_EACCESSDENIED	Transport already open.

Pre Conditions

mqtAttrs must be valid.

Post Conditions

None

See Also

MSGQ_TransportClose

6.4 MSGQ_TransportClose

This function finalizes the MQT component.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_TransportClose (IN  TransportId mqtId) ;
```

Arguments

IN	TransportId	mqtId
	ID of the MQT to be closed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.
DSP_EACCESSDENIED	Not the owner.

Pre Conditions

None

Post Conditions

None

See Also

MSGQ_TransportOpen

6.5 MSGQ_Create

This function creates the message queue to be used for receiving messages, identified through the specified MSGQ ID.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Create (IN  MsgQueueId msgqId, IN OPT MsgqAttrs * msgqAttrs) ;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the message queue to be created.	
IN OPT	MsgqAttrs *	msgqAttrs
	Optional attributes for creation of the MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

msgqId must be valid.

Post Conditions

None

See Also

MSGQ_Delete
MSGQ_Locate

6.6 MSGQ_Delete

This function deletes the message queue identified by the specified MSGQ ID.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Delete (IN  MsgQueueId msgqId) ;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the message queue to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

msgqId must be valid.

Post Conditions

None

See Also

MSGQ_Create

6.7 MSGQ_Locate

This function verifies the existence and status of the message queue identified by the specified MSGQ ID, on the specified processor.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Locate (IN  ProcessorId       procId,
             IN  MsgQueueId        msgqId,
             IN  MsgqLocateAttrs * attrs) ;
```

Arguments

IN	ProcessorId	procId
	Processor identifier.	
IN	MsgQueueId	msgqId
	ID of the MQT to be opened.	
IN	MsgqLocateAttrs *	attrs
	Optional attributes for location of the MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_ETIMEOUT	Timeout occurred while locating the message.
DSP_ENOTFOUND	The message queue does not exist on the specified processor.
DSP_ENOTCOMPLETE	
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

procId must be valid.

msgqId must be valid.

attrs must be valid.

Post Conditions

None

See Also

MsgQueueId
MSGQ_Put
MSGQ_Release

6.8 MSGQ_Release

This function releases the MSGQ located through an earlier MSGQ_Locate () or MSGQ_GetReplyId () call.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Release (IN  ProcessorId procId, IN  MsgQueueId msgqId) ;
```

Arguments

IN	ProcessorId	procId
	Processor identifier.	
IN	MsgQueueId	msgqId
	ID of the MQT to be opened.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_ENOTFOUND	The message queue does not exist on the specified processor. This error occurs when the MSGQ_Locate was not called / was unsuccessful for the msgqId on the processor specified.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

procId must be valid.

msgqId must be valid.

Post Conditions

None

See Also

MsgQueueId
MSGQ_Locate

6.9 MSGQ_Alloc

This function allocates a message, and returns the pointer to the user.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Alloc (IN  AllocatorId mqaId, IN  Uint16 size, OUT MsgqMsg * msg)
;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be used for allocating this message.	
IN	Uint16	size
	Size of the message to be allocated.	
OUT	MsgqMsg *	msg
	Location to receive the allocated message.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

msg must be valid.

size must be greater than size of MsgqMsgHeader.

Post Conditions

None

See Also

MsgqMsgHeader
MSGQ_Put

6.10 MSGQ_Free

This function frees a message.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Free (IN  MsgqMsg msg) ;
```

Arguments

IN	MsgqMsg	msg
	Pointer to the message to be freed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

msg must be valid.

Post Conditions

None

See Also

MsgqMsgHeader
MSGQ_Get

6.11 MSGQ_Put

This function sends a message to the specified MSGQ on the specified processor.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Put (IN     ProcessorId procId,
          IN     MsgQueueId  destMsgqId,
          IN     MsgqMsg     msg,
          IN OPT Uint16      msgId,
          IN OPT MsgQueueId  srcMsgqId) ;
```

Arguments

IN	ProcessorId	procId
	Processor identifier on which the MSGQ exists.	
IN	MsgQueueId	destMsgqId
	ID of the destination MSGQ.	
IN	MsgqMsg	msg
	Pointer to the message to be sent to the destination MSGQ.	
IN OPT	Uint16	msgId
	Optional message ID to be associated with the message.	
IN OPT	MsgQueueId	srcMsgqId
	Optional ID of the source MSGQ to receive reply messages.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_ENOTFOUND	The message queue does not exist on the specified processor.

Pre Conditions

procId must be valid.

msg must be valid.

destMsgqId must be valid.

Post Conditions

None

See Also

MsgqMsgHeader
MSGQ_Get

6.12 MSGQ_Get

This function receives a message on the specified MSGQ.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Get (IN  MsgQueueId msgqId, IN  Uint32 timeout, OUT MsgqMsg * msg)
;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the MSGQ on which the message is to be received.	
IN	Uint32	timeout
	Timeout value to wait for the message (in milliseconds).	
OUT	MsgqMsg *	msg
	Location to receive the message.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_ETIMEOUT	Timeout occurred while receiving the message.
DSP_ENOTCOMPLETE	
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

msg must be valid.

msgqId must be valid.

Post Conditions

None

See Also

MsgqMsgHeader
MSGQ_Put

6.13 MSGQ_GetReplyId

This function extracts the MSGQ ID and processor ID to be used for replying to a received message.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_GetReplyId (IN  MsgqMsg       msg,
                 OUT ProcessorId * procId,
                 OUT MsgQueueId *  msgqId) ;
```

Arguments

IN	MsgqMsg	msg
	Message, whose reply MSGQ ID is to be extracted.	
OUT	ProcessorId *	procId
	Location to retrieve the ID of the processor where the reply MSGQ resides.	
OUT	MsgQueueId *	msgqId
	Location to retrieve the ID of the reply MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

msg must be valid.

procId must be valid.

msgqId must be valid.

Post Conditions

None

See Also

MsgqMsgHeader

6.14 MSGQ_SetErrorHandler

This API allows the user to designate a MSGQ as an error-handler MSGQ to receive asynchronous error messages from the transports.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_SetErrorHandler (IN  MsgQueueId msgqId, IN  Uint16 mqaId) ;
```

Arguments

IN	MsgQueueId	msgqId
	Message queue to receive the error messages.	
IN	Uint16	mqaId
	ID indicating the allocator to be used for allocating the error messages.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid Parameter passed.
DSP_EFAIL	General failure.

Pre Conditions

msgqId must be valid.

Post Conditions

None

See Also

MsgqAsyncErrorMsg

6.15 MSGQ_Instrument

This function gets the instrumentation information related to the specified message queue.

Syntax

```
EXPORT_API
DSP_STATUS
MSGQ_Instrument (IN  ProcessorId      procId,
                 IN  MsgQueueId       msgqId,
                 OUT MsgqInstrument * retVal) ;
```

Arguments

IN	ProcessorId	procId
	Processor identifier.	
IN	MsgQueueId	msgqId
	Message queue identifier.	
OUT	MsgqInstrument *	retVal
	Location to retrieve the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

procId must be valid.

msgqId must be valid.

retVal must be valid.

Post Conditions

None

See Also

MsgqInstrument

6.16 MSGQ_Debug

This function prints the current status of the MSGQ subcomponent.

Syntax

```
EXPORT_API
Void
MSGQ_Debug (IN  ProcessorId procId, IN  MsgQueueId msgqId) ;
```

Arguments

IN	ProcessorId	procId
	Processor identifier.	
IN	MsgQueueId	msgqId
	Message queue identifier.	

Return Values

None

Pre Conditions

procId must be valid.

msgqId must be valid.

Post Conditions

None

See Also

None

7 msgqdefs.h

Defines data types and structures used by DSP/BIOS(tm) Link for MSGQ.

Path

$(DSPLINK)\gpp\inc\Linux

Revision

00.03

CONSTANTS

7.1 MAX_ALLOCATORS

Maximum number of allocators supported by DSP/BIOS Link.

Syntax

```
#define MAX_ALLOCATORS    16
```

See Also

7.2 MAX_MSGQS

Maximum number of message queues that can be created on the GPP.

Syntax

```
#define MAX_MSGQS    32
```

See Also

7.3 ID_LOCAL_PROCESSOR

This constant defines the ID of the local processor (GPP).

Syntax

```
#define ID_LOCAL_PROCESSOR    (ProcessorId) 0xFFFF
```

See Also

7.4 MSGQ_INTERNAL_ID_START

This constant defines the start of internal MSGQ message id range.

Syntax

```
#define MSGQ_INTERNAL_ID_START (Uint16) 0xFE00
```

See Also

7.5 MSGQ_ASYNC_ERROR_MSGID

This constant defines the asynchronous error message id.

Syntax

```
#define MSGQ_ASYNC_ERROR_MSGID (Uint16) 0xFE01
```

See Also

7.6 MSGQ_INTERNAL_ID_END

This constant defines the end of internal MSGQ message id range.

Syntax

```
#define MSGQ_INTERNAL_ID_END (Uint16) 0xFEFF
```

See Also

7.7 MSGQ_MQT_MSGID_START

This constant defines the start of transport message id range.

Syntax

```
#define MSGQ_MQT_MSGID_START (Uint16) 0xFF00
```

See Also

7.8 MSGQ_MQT_MSGID_END

This constant defines the end of transport message id range.

Syntax

```
#define MSGQ_MQT_MSGID_END (Uint16) 0xFFFE
```

See Also

7.9 MSGQ_INVALID_ID

This constant defines the invalid ID for MSGQ, MQT, MQA and messages.

Syntax

```
#define MSGQ_INVALID_ID   (Uint16) 0xFFFF
```

See Also

7.10 MSG_HEADER_RESERVED_SIZE

This macro defines the size of the reserved field of message header.

Syntax

```
#define MSG_HEADER_RESERVED_SIZE    2
```

See Also

7.11 MSGQ_GetMsgId

This macro returns the message ID of the specified message.

Syntax

```
#define MSGQ_GetMsgId(msg) (((MsgqMsg) (msg))->msgId)
```

See Also

7.12 MSGQ_GetMsgSize

This macro returns the size of the specified message.

Syntax

```
#define MSGQ_GetMsgSize(msg) (((MsgqMsg) (msg))->size)
```

See Also

ENUMERATIONS

7.13 MsgqErrorType

This enumeration defines the possible types of asynchronous error messages.

Syntax

```
typedef enum {
    MsgqErrorType_MqtExit   = 0,
    MsgqErrorType_PutFailed = 1
} MsgqErrorType ;
```

Enum Values

MsgqErrorType_MqtExit	Indicates that remote MQT has called exit. The arg1 and arg2 field of MsgqAsyncErrorMsg structure shall be as follows in this error type : arg1 : Mqt Id of the remote MSGQ. arg2 : Not used. arg3 : Not used.
MsgqErrorType_PutFailed	Indicates that MSGQ_Put failed. The arg1 and arg2 field of MsgqAsyncErrorMsg structure shall be as follows in this error type : arg1 : ID of the processor on which the destination message queue exists. arg2 : ID of the destination message queue on which the put failed. arg3 : Status of the MSGQ_Put call that failed.

See Also

MsgqAsyncErrorMsg

TYPE DEFINITIONS & STRUCTURES

7.14 MsgqMsgHeader

This structure defines the format of the message header that must be the first field of any message.

Syntax

```
typedef struct MsgqMsgHeader_tag {
    Uint32    reserved [MSG_HEADER_RESERVED_SIZE] ;
    Uint16    mqtId        ;
    Uint16    mqaId        ;
    Uint16    size         ;
    Uint16    dstId        ;
    Uint16    srcId        ;
    Uint16    msgId        ;
} MsgqMsgHeader ;
```

Fields

Uint32	reserved Reserved for use by the MQT. The MQT typically uses them as a link for queuing the messages.
Uint16	mqtId ID of the MQT used for transporting this message.
Uint16	mqaId ID of the MQA used for allocating this message.
Uint16	size Size of the message including the header.
Uint16	dstId ID of the destination message queue.
Uint16	srcId ID of the source message queue for reply.
Uint16	msgId User-specified message ID.

See Also

PROCESSOR MANAGER COMPONENT

8 pmgr_proc.h

Defines the interfaces and data structures for the sub-component PMGR_PROC.

Path

`$(DSPLINK)\gpp\src\pmgr`

Revision

`00.10`

FUNCTIONS

8.1 PMGR_PROC_Setup

Sets up the necessary data structures for the PMGR_PROC sub-component.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Setup () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EACCESSDENIED	Access denied. Another client has already called PMGR_PROC_Setup ().
DSP_EFAIL	General failure.

Pre Conditions

None

Post Conditions

None

See Also

PMGR_PROC_Destroy

8.2 PMGR_PROC_Destroy

Destroys the data structures for the PMGR_PROC component, allocated earlier by a call to PROC_Setup ().

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Destroy () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EACCESSDENIED	Access denied. Only the client who had successfully called PMGR_PROC_Setup () can call this function.
DSP_EFAIL	General failure.

Pre Conditions

None

Post Conditions

None

See Also

PMGR_PROC_Setup

8.3 PMGR_PROC_Attach

Attaches the client to the specified DSP and also initializes the DSP (if required).

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Attach (IN    ProcessorId   procId,
                  OPT   ProcAttr *    attr) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
OPT	ProcAttr *	attr
	Attributes for the processor on which attach is to be done.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_SALREADYATTACHED	Successful attach. Also, indicates that another client has already attached to DSP.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EFAIL	General failure, unable to attach to processor.
DSP_EWRONGSTATE	Incorrect state to complete the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PMGR_PROC_Detach

8.4 PMGR_PROC_Detach

Detaches the client from specified processor.

If the caller is the owner of the processor, this function releases all the resources that this component uses and puts the DSP in an unusable state (from application perspective).

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Detach (IN  ProcessorId   procId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure, unable to detach.
DSP_EACCESSDENIED	Not allowed to access the DSP.
DSP_EATTACHED	Not attached to the target processor.
DSP_EWRONGSTATE	Incorrect state to complete the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PMGR_PROC_Attach

8.5 PMGR_PROC_GetState

Gets the current status of DSP by querying the Link Driver.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_GetState (IN   ProcessorId   procId,
                    OUT  ProcState *   procState) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
OUT	ProcState *	procState
	Placeholder for processor state.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure, couldn't get state information.

Pre Conditions

procId must be valid.

The pointer to ProcState is valid.

Post Conditions

None

See Also

PMGR_PROC_Load
PMGR_PROC_Start
PMGR_PROC_Stop
PMGR_PROC_Idle

8.6 PMGR_PROC_Load

Loads the specified DSP executable on the target DSP.

It ensures that the caller owns the DSP.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Load (IN ProcessorId  procId,
                IN Char8 *      imagePath,
                IN Uint32       argc,
                IN Char8 **     argv) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	Char8 *	imagePath
	Full path to the image file to load on DSP.	
IN	Uint32	argc
	Number of arguments to be passed to the DSP executable.	
IN	Char8 **	argv
	Arguments to be passed to DSP executable.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EACCESSDENIED	Access denied. Only the owner client is allowed to load the DSP executable.
DSP_EFILE	Invalid DSP executable.
DSP_EFAIL	General failure, unable to load DSP executable.
DSP_EWRONGSTATE	Incorrect state to complete the requested operation.

Pre Conditions

procId must be valid.

Base image path must be valid.

If argc is 0 then argv must be NULL pointer.

If argc is non-zero then argv must be a valid pointer.

Post Conditions

None

See Also

PMGR_PROC_Attach
PMGR_PROC_LoadSection

8.7 PMGR_PROC_LoadSection

Loads the specified section of DSP executable onto the target DSP.

It ensures that the client owns the DSP.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_LoadSection (IN ProcessorId  procId,
                       IN Char8 *      imagePath,
                       IN Uint32       sectID) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	Char8 *	imagePath
	Full path to the image file.	
IN	Uint32	sectID
	Section ID of section to load.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFILE	Invalid ImagePath parameter.
DSP_EINVALIDSECTION	Invalid section id.
DSP_EACCESSDENIED	Access denied. Only the owner client is allowed to load section on the DSP.
DSP_EFAIL	General failure, unable to load section on DSP.
DSP_EWRONGSTATE	Incorrect state to complete the requested operation.

Pre Conditions

procId must be valid.

Base image path must be valid.

Post Conditions

None

See Also

PMGR_PROC_Attach

PMGR_PROC_Load

8.8 PMGR_PROC_Start

Starts execution of the loaded code on DSP from the starting point specified in the DSP executable loaded earlier by call to PROC_Load ().

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Start (IN  ProcessorId  procId) ;
```

Arguments

IN	ProcessorId	procId
	DSP Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EACCESSDENIED	Access denied. Only the owner client is allowed to start the DSP.
DSP_EFAIL	General failure, unable to start DSP.
DSP_EWRONGSTATE	Incorrect state to complete the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PMGR_PROC_Attach
PMGR_PROC_Load
PMGR_PROC_Stop

8.9 PMGR_PROC_Stop

Stops the specified DSP.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Stop (IN   ProcessorId  procId) ;
```

Arguments

IN	ProcessorId	procId
	DSP Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EACCESSDENIED	Access denied. Only the owner client is allowed to stop the DSP.
DSP_EFAIL	General failure, unable to stop DSP.
DSP_EWRONGSTATE	Incorrect state to complete the requested operation.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PMGR_PROC_Attach
PMGR_PROC_Load
PMGR_PROC_Start

8.10 PMGR_PROC_Control

Provides a hook to perform device dependent control operations.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Control (IN  ProcessorId procId,
                   IN  Int32       cmd,
                   OPT Pvoid       arg) ;
```

Arguments

IN	ProcessorId	procId
	DSP Identifier.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PMGR_PROC_Attach

8.11 PMGR_PROC_IsAttached

Function to check whether the client identified by the specified 'client' object is attached to the specified processor.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_IsAttached (IN  ProcessorId  procId,
                      IN  PrcsObject * client,
                      OUT Bool *       isAttached) ;
```

Arguments

IN	ProcessorId	procId
	Processor Id.	
IN	PrcsObject *	client
	Client identifier.	
OUT	Bool *	isAttached
	Place holder for flag indicating the client is attached.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

PMGR_PROC_Attach

8.12 PMGR_PROC_Instrument

Gets the instrumentation data associated with PMGR_PROC sub-component.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_PROC_Instrument (IN ProcessorId procId, OUT ProcInstrument *
retVal) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor for which instrumentation information is to be obtained.	
OUT	ProcInstrument *	retVal
	OUT argument to contain the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	retVal is invalid.

Pre Conditions

procId must be valid.

retVal must be a valid pointer.

Post Conditions

None

See Also

None

8.13 PMGR_PROC_Debug

Prints the debug information summarizing the current status of the PMGR_PROC component.

Syntax

```
NORMAL_API
Void
PMGR_PROC_Debug (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId	Identifier for processor.

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

9 pmgr_chnl.h

Defines the interfaces and data structures for the sub-component PMGR_CHNL.

Path

$(DSPLINK)\gpp\src\pmgr

Revision

00.08

FUNCTIONS

9.1 PMGR_CHNL_Initialize

Sets up all channel objects in Link Driver.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Initialize (IN ProcessorId    procId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Id.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Processor Id must be valid.

Post Conditions

None

See Also

PMGR_CHNL_Finalize
PMGR_CHNL_Create

9.2 PMGR_CHNL_Finalize

Releases all channel objects setup in Link Driver.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Finalize (IN ProcessorId    procId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Id.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processor must be initialized.

Processor Id must be valid.

Post Conditions

None

See Also

```
PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_Destroy
```

9.3 PMGR_CHNL_Create

Creates resources used for transferring data between GPP and DSP.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Create (IN ProcessorId    procId,
                  IN ChannelId      chnlId,
                  IN ChannelAttrs * attrs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Id to open.	
IN	ChannelAttrs *	attrs
	Channel attributes.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

Attributes must be valid.

Post Conditions

None

See Also

PMGR_CHNL_Initialize

9.4 PMGR_CHNL_Delete

Releases channel resources used for transferring data between GPP and DSP.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Delete (IN ProcessorId    procId,
                  IN ChannelId      chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processors should have been initialized.

Processor and channel ids should be valid.

Post Conditions

None

See Also

PMGR_CHNL_Create

9.5 PMGR_CHNL_AllocateBuffer

Allocates an array of buffers of specified size and returns them to the client.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_AllocateBuffer (IN  ProcessorId procId,
                          IN  ChannelId   chnlId,
                          OUT Char8 **    bufArray,
                          IN  Uint32      size,
                          IN  Uint32      numBufs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
OUT	Char8 **	bufArray
	Pointer to receive array of allocated buffers.	
IN	Uint32	size
	Size of each buffer.	
IN	Uint32	numBufs
	Number of buffers to allocate.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

bufArray must be valid.

numBufs must be less than maximum limit.

Post Conditions

None

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_FreeBuffer

9.6 PMGR_CHNL_FreeBuffer

Frees buffer(s) allocated by PMGR_CHNL_AllocateBuffer.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_FreeBuffer (IN ProcessorId procId,
                      IN ChannelId   chnlId,
                      IN Char8 **    bufArray,
                      IN Uint32      numBufs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Char8 **	bufArray
	Pointer to the array of buffers to freed.	
IN	Uint32	numBufs
	Number of buffers to be freed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

bufArray must be valid.

numBufs must be less than maximum limit.

Post Conditions

None

See Also

PMGR_CHNL_Initialize

```
PMGR_CHNL_Create
PMGR_CHNL_AllocateBuffer
```

9.7 PMGR_CHNL_Issue

Issues an input or output request on a specified channel.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Issue (IN ProcessorId      procId,
                 IN ChannelId        chnlId,
                 IN ChannelIOInfo *  ioReq) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	ChannelIOInfo *	ioReq
	Information regarding IO.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

ioReq must be valid pointer.

Post Conditions

None

See Also

PMGR_CHNL_Reclaim

9.8 PMGR_CHNL_Reclaim

Gets the buffer back that has been issued to this channel.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Reclaim (IN     ProcessorId       procId,
                   IN     ChannelId         chnlId,
                   IN     Uint32            timeout,
                   IN OUT ChannelIOInfo *   ioReq) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Uint32	timeout
	Timeout value for this operation.	
IN OUT	ChannelIOInfo *	ioReq
	Information needed for doing reclaim.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.
CHNL_E_NOIOC	Timeout parameter was "NO_WAIT", yet no I/O completions were queued.

Pre Conditions

Channels for specified processors must be initialized.

Processor and channel ids must be valid.

ioReq must be valid pointer.

Post Conditions

None

See Also

PMGR_CHNL_Issue

9.9 PMGR_CHNL_Idle

If the channel is an input stream this function resets the channel and causes any currently buffered input data to be discarded. If the channel is an output channel, this function causes any currently queued buffers to be transferred through the channel. It causes the client to wait for as long as it takes for the data to be transferred through the channel.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Idle (IN ProcessorId  procId,
                IN ChannelId    chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create

9.10 PMGR_CHNL_Flush

Discards all the requested buffers that are pending for transfer both in case of input mode channel as well as output mode channel.

One must still have to call the PMGR_CHNL_Reclaim to get back the discarded buffers.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Flush (IN ProcessorId         procId,
                 IN ChannelId           chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_Issue

9.11 PMGR_CHNL_Control

Provides a hook to perform device dependent control operations on channels.

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Control (IN  ProcessorId    procId,
                   IN  ChannelId      chnlId,
                   IN  Int32          cmd,
                   OPT Pvoid          arg) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

PMGR_CHNL_Initialize

9.12 PMGR_CHNL_Instrument

Gets the instrumentation information related to CHNL's

Syntax

```
NORMAL_API
DSP_STATUS
PMGR_CHNL_Instrument (IN  ProcessorId       procId,
                      IN  ChannelId         chnlId,
                      OUT ChnlInstrument *  retVal) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor.	
IN	ChannelId	chnlId
	Identifier for channel for which instrumentation information is to be obtained.	
OUT	ChnlInstrument *	retVal
	OUT argument to contain the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	retVal is invalid.

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

retVal must be a valid pointer.

Post Conditions

None

See Also

None

9.13 PMGR_CHNL_Debug

Prints the current status of CHNL subcomponent.

Syntax

```
NORMAL_API
Void
PMGR_CHNL_Debug (ProcessorId procId, ChannelId chnlId) ;
```

Arguments

ProcessorId	procId
Processor Identifier.	
ChannelId	chnlId
Channel Identifier.	

Return Values

None

Pre Conditions

Channels for specified processor must be initialized.

Processor and channel ids must be valid.

Post Conditions

None

See Also

PMGR_CHNL_Initialize

10 pmgr_msgq.h

Defines the interfaces and data structures for the sub-component PMGR_MSGQ.

Path

$(DSPLINK)\gpp\src\pmgr

Revision

00.05

FUNCTIONS

10.1 PMGR_MSGQ_Setup

This function initializes the MSGQ component.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Setup () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must not be initialized before calling this function.

Post Conditions

None

See Also

LDRV_MSGQ_Setup

10.2 PMGR_MSGQ_Destroy

This function finalizes the MSGQ component.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Destroy () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

Post Conditions

None

See Also

LDRV_MSGQ_Destroys

10.3 PMGR_MSGQ_AllocatorOpen

This function initializes the MQA component.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_AllocatorOpen (IN  AllocatorId mqaId,
                         IN  Pvoid       mqaAttrs,
                         OUT Pvoid *     mqaInfo) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be opened.	
IN	Pvoid	mqaAttrs
	Attributes for initialization of the MQA component. The structure of the expected attributes is specific to every MQA.	
OUT	Pvoid *	mqaInfo
	Location to receive the handle to the initialized MQA state object.	

Return Values

DSP_SOK	The MQA component has been successfully opened.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid Parameter passed.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

mqaAttrs must be valid.

mqaInfo must be valid.

Post Conditions

None

See Also

MSGQ_AllocatorOpen
LDRV_MSGQ_AllocatorOpen

10.4 PMGR_MSGQ_AllocatorClose

This function finalizes the allocator component.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_AllocatorClose (IN  AllocatorId mqaId) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be closed.	

Return Values

DSP_SOK	The MQA component has been successfully closed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

Post Conditions

None

See Also

MSGQ_AllocatorClose
LDRV_MSGQ_AllocatorClose

10.5 PMGR_MSGQ_TransportOpen

This function initializes the MQT component.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_TransportOpen (IN  TransportId mqtId, IN  Pvoid mqtAttrs) ;
```

Arguments

IN	TransportId	mqtId
	ID of the MQT to be opened.	
IN	Pvoid	mqtAttrs
	Attributes for initialization of the MQT component. The structure of the expected attributes is specific to every MQT.	

Return Values

DSP_SOK	The MQT component has been successfully opened.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

mqtAttrs must be valid.

PMGR MSGQ component must be initialized before calling this function.

Post Conditions

None

See Also

PMGR_MSGQ_TransportClose

10.6 PMGR_MSGQ_TransportClose

This function finalizes the MQT component.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_TransportClose (IN  TransportId mqtId) ;
```

Arguments

IN	TransportId	mqtId
	ID of the MQT to be closed.	

Return Values

DSP_SOK	The MQT component has been successfully closed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

Post Conditions

None

See Also

PMGR_MSGQ_TransportOpen

10.7 PMGR_MSGQ_Create

This function creates the message queue to be used for receiving messages, identified through the specified MSGQ ID.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Create (IN  MsgQueueId msgqId, IN OPT MsgqAttrs * msgqAttrs)
;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the message queue to be created.	
IN OPT	MsgqAttrs *	msgqAttrs
	Optional attributes for creation of the MSGQ.	

Return Values

DSP_SOK	The message queue has been successfully created.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

msgqId must be valid.

Post Conditions

None

See Also

PMGR_MSGQ_Delete
PMGR_MSGQ_Locate

10.8 PMGR_MSGQ_Delete

This function deletes the message queue identified by the specified MSGQ ID.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Delete (IN  MsgQueueId msgqId) ;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the message queue to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

msgqId must be valid.

Client should be the owner of msgqId MSGQ.

Post Conditions

None

See Also

PMGR_MSGQ_Create

10.9 PMGR_MSGQ_Locate

This function verifies the existence and status of the message queue identified by the specified MSGQ ID, on the specified processor.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Locate (IN  ProcessorId       procId,
                  IN  MsgQueueId        msgqId,
                  IN  MsgqLocateAttrs * attrs) ;
```

Arguments

IN	ProcessorId	procId
	ID of the processor on which the MSGQ is to be located.	
IN	MsgQueueId	msgqId
	ID of the message queue to be located.	
IN	MsgqLocateAttrs *	attrs
	Attributes for location of the MSGQ.	

Return Values

DSP_SOK	The message queue has been successfully located.
DSP_EMEMORY	Operation failed due to memory error.
DSP_ETIMEOUT	Timeout occurred while locating the MSGQ.
DSP_EFAIL	General failure.
DSP_ENOTFOUND	The message queue does not exist on the specified processor.

Pre Conditions

procId must be valid.

PMGR MSGQ component must be initialized before calling this function.

msgqId must be valid.

attrs must be valid.

Post Conditions

None

See Also

MsgQueueId
PMGR_MSGQ_Put
PMGR_MSGQ_Release

10.10 PMGR_MSGQ_Release

This function releases the MSGQ located earlier.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Release (IN  ProcessorId procId, IN  MsgQueueId msgqId) ;
```

Arguments

IN	ProcessorId	procId
	Processor identifier.	
IN	MsgQueueId	msgqId
	ID of the message queue to be released.	

Return Values

DSP_SOK	The message queue has been successfully released.
DSP_EFAIL	General failure.
DSP_ENOTFOUND	The message queue does not exist on the specified processor. This error occurs when the MSGQ_Locate was not called / was unsuccessful for the msgqId on the processor specified.

Pre Conditions

procId must be valid.

PMGR MSGQ component must be initialized before calling this function.

msgqId must be valid.

Post Conditions

None

See Also

MsgQueueId
PMGR_MSGQ_Locate

10.11 PMGR_MSGQ_Alloc

This function allocates a message, and returns the pointer to the user.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Alloc (IN  AllocatorId mqaId, IN  Uint16 size, OUT MsgqMsg  *
msg) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be used for allocating this message.	
IN	Uint16	size
	Size of the message to be allocated.	
OUT	MsgqMsg *	msg
	Location to receive the allocated message.	

Return Values

DSP_SOK	The message has been successfully allocated.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

msg must be valid.

Post Conditions

None

See Also

MsgqMsgHeader
PMGR_MSGQ_Put

10.12 PMGR_MSGQ_Free

This function frees a message.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Free (IN  MsgqMsg   msg) ;
```

Arguments

IN	MsgqMsg	msg
	Pointer to the message to be freed.	

Return Values

DSP_SOK	The message has been successfully freed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

msg must be valid.

Post Conditions

None

See Also

MsgqMsgHeader
PMGR_MSGQ_Get

10.13 PMGR_MSGQ_Put

This function sends a message to the specified MSGQ on a particular processor.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Put (IN     ProcessorId procId,
               IN     MsgQueueId  destMsgqId,
               IN     MsgqMsg     msg,
               IN OPT Uint16      msgId,
               IN OPT MsgQueueId  srcMsgqId) ;
```

Arguments

IN	ProcessorId	procId
	ID of the processor on which the MSGQ to which the message is to be sent, exists.	
IN	MsgQueueId	destMsgqId
	ID of the destination MSGQ.	
IN	MsgqMsg	msg
	Pointer to the message to be sent to the destination MSGQ.	
IN OPT	Uint16	msgId
	Optional message ID to be associated with the message.	
IN OPT	MsgQueueId	srcMsgqId
	Optional ID of the source MSGQ to receive reply messages.	

Return Values

DSP_SOK	The message has been successfully sent.
DSP_EFAIL	General failure.
DSP_ENOTFOUND	The message queue does not exist. This implies that the MSGQ has not been located before this call was made.

Pre Conditions

procId must be valid.

PMGR MSGQ component must be initialized before calling this function.

msg must be valid.

destMsgqId must be valid.

Post Conditions

None

See Also

MsgqMsgHeader
PMGR_MSGQ_Get

10.14 PMGR_MSGQ_Get

This function receives a message on the specified MSGQ.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Get (IN  MsgQueueId msgqId, IN  Uint32 timeout, OUT MsgqMsg *
msg) ;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the MSGQ on which the message is to be received.	
IN	Uint32	timeout
	Timeout value to wait for the message (in milliseconds).	
OUT	MsgqMsg *	msg
	Location to receive the message.	

Return Values

DSP_SOK	The message has been successfully received.
DSP_EFAIL	General failure.
DSP_ETIMEOUT	Timeout occurred while receiving the message.

Pre Conditions

PMGR MSGQ component must be initialized before calling this function.

msg must be valid.

msgqId must be valid.

Client should be the owner of msgqId MSGQ.

Post Conditions

None

See Also

MsgqMsgHeader
PMGR_MSGQ_Put

10.15 PMGR_MSGQ_GetReplyId

This function extracts the MSGQ ID and processor ID to be used for replying to a received message.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_GetReplyId (IN  MsgqMsg       msg,
                      OUT ProcessorId * procId,
                      OUT MsgQueueId *  msgqId) ;
```

Arguments

IN	MsgqMsg	msg
	Message, whose reply MSGQ ID is to be extracted.	
OUT	ProcessorId *	procId
	Location to retrieve the ID of the processor where the reply MSGQ resides.	
OUT	MsgQueueId *	msgqId
	Location to retrieve the ID of the reply MSGQ.	

Return Values

DSP_SOK	The reply information has been successfully retrieved.
DSP_ENOTFOUND	Reply information has not been provided by the sender.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

msg must be valid.

procId must be valid.

msgqId must be valid.

PMGR MSGQ component must be initialized before calling this function.

Post Conditions

None

See Also

MSGQ_GetReplyId

```
LDRV_MSGQ_GetReplyId
```

10.16 PMGR_MSGQ_SetErrorHandler

This function allows the user to designate a MSGQ as an error-handler MSGQ to receive asynchronous error messages from the transports.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_SetErrorHandler (IN  MsgQueueId msgqId, OUT Uint16 mqaId) ;
```

Arguments

IN	MsgQueueId	msgqId
	Message queue to receive the error messages.	
OUT	Uint16	mqaId
	ID indicating the allocator to be used for allocating the error messages.	

Return Values

DSP_SOK	The error handler has been successfully set.
DSP_EFAIL	General failure.

Pre Conditions

msgqId must be valid.

Post Conditions

None

See Also

MSGQ_SetErrorHandler
LDRV_MSGQ_SetErrorHandler

10.17 PMGR_MSGQ_Instrument

This function gets the instrumentation information related to the specified message queue.

Syntax

```
EXPORT_API
DSP_STATUS
PMGR_MSGQ_Instrument (IN  ProcessorId      procId,
                      IN  MsgQueueId       msgqId,
                      OUT MsgqInstrument * retVal) ;
```

Arguments

IN	ProcessorId	procId
	Processor identifier.	
IN	MsgQueueId	msgqId
	Message queue identifier.	
OUT	MsgqInstrument *	retVal
	Location to retrieve the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

msgqId must be valid.

PMGR MSGQ component must be initialized before calling this function.

retVal must be valid.

Post Conditions

None

See Also

MsgqInstrument

10.18 PMGR_MSGQ_Debug

This function prints the current status of the MSGQ subcomponent.

Syntax

```
EXPORT_API
Void
PMGR_MSGQ_Debug (IN  ProcessorId procId, IN  MsgQueueId msgqId) ;
```

Arguments

IN	ProcessorId	procId
	ID of the MSGQ on which the message is to be received.	
IN	MsgQueueId	msgqId
	Timeout value to wait for the message (in milliseconds).	

Return Values

None

Pre Conditions

procId must be valid.

msgqId must be valid.

PMGR MSGQ component must be initialized before calling this function.

Post Conditions

None

See Also

None

LINK DRIVER COMPONENT

11 linkdefs.h

Definitions of constants and structures for Link.

Path

$(DSPLINK)\gpp\inc

Revision

00.11

TYPE DEFINITIONS & STRUCTURES

11.1 LinkAttrs_tag

Defines the attributes of a link.

Syntax

```
struct LinkAttrs_tag {
    Char8   linkName [DSP_MAX_STRLEN] ;
    Char8   abbr [DSP_MAX_STRLEN] ;
    Uint32  baseChnlId  ;
    Uint32  numChannels ;
    Uint32  maxBufSize  ;
    Void *  interface   ;
    Uint32  argument1   ;
    Uint32  reserved2   ;
} ;
```

Fields

Char8	linkName Name of physical link.
Char8	abbr Abbreviation for the driver name.
Uint32	baseChnlId Start channel Id for the link.
Uint32	numChannels Number of (virtual) channels on link.
Uint32	maxBufSize Maximum buffer size supported on channel
Void *	interface The table of function pointers from DSP subcomponent.
Uint32	argument1

	Link specific argument 1. The significance of this argument is specific to a link driver.
Uint32	reserved2 Link reserved argument 2. The significance of this argument is specific to a link driver.

See Also

FUNCTIONS

11.2 FnLinkInitialize

Signature of function that allocates and initializes resources used by the Link Driver.

Syntax

```
typedef DSP_STATUS (*FnLinkInitialize) (IN ProcessorId dspId,
                                        IN LinkId      linkId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	LinkId	linkId
	Identifier of link to be initialized.	

Return Values

DSP_SOK	Operation completed successfully.
DSP_EMEMORY	Out of memory

Pre Conditions

Post Conditions

See Also

11.3 FnLinkFinalize

Signature of function that de-allocates and finalizes resources used by the Shared Memory Driver.

Syntax

```
typedef DSP_STATUS (*FnLinkFinalize) (IN ProcessorId dspId,
                                      IN LinkId      linkId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	LinkId	linkId
	Identifier of link to be finalized.	

Return Values

DSP_SOK	Operation completed successfully.

Pre Conditions

Post Conditions

See Also

11.4 FnLinkOpenChannel

Signature of function that opens a channel for input/output.

Syntax

```
typedef DSP_STATUS (*FnLinkOpenChannel) (IN ProcessorId dspId,
                                         IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	ChannelId	chnlId
	Channel ID on which IO is being requested.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	Could not open the channel successfully.

Pre Conditions

Post Conditions

See Also

11.5 FnLinkCloseChannel

Signature of function that closes a channel.

Syntax

```
typedef DSP_STATUS (*FnLinkCloseChannel) (IN ProcessorId dspId,
                                          IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	ChannelId	chnlId
	Channel ID on which IO is being requested.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	Could not open the channel successfully.

Pre Conditions

Post Conditions

See Also

11.6 FnLinkCancelIO

Signature of function that cancels a channel.

Syntax

```
typedef DSP_STATUS (*FnLinkCancelIO) (IN ProcessorId dspId,
                                      IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	Could not open the channel successfully.

Pre Conditions

Post Conditions

See Also

11.7 FnLinkIORequest

Signature of function that de-allocates and finalizes resources used by the Shared Memory Driver.

Syntax

```
typedef DSP_STATUS (*FnLinkIORequest) (IN ProcessorId dspId,
                                       IN ChannelId   chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	ChannelId	chnlId
	Channel ID on which IO is being requested.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	chnlId is invalid.
DSP_EPOINTER	Subcomponent is not initialized.

Pre Conditions

Post Conditions

See Also

11.8 FnLinkScheduleDpc

Signature of function that schedules a DPC for IO on the specified channel.

Syntax

```
typedef DSP_STATUS (*FnLinkScheduleDpc) (IN ProcessorId dspId,
                                         IN ChannelId   chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	ChannelId	chnlId
	Channel ID on which IO is being requested.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

Post Conditions

See Also

11.9 FnLinkHandshakeSetup

Signature of function that setup the handshake process toward specified DSP on the Link Driver.

Syntax

```
typedef DSP_STATUS (*FnLinkHandshakeSetup) (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

Post Conditions

See Also

11.10 FnLinkHandshakeStart

Signature of function that starts the handshake process toward specified DSP on the Link Driver.

Syntax

```
typedef DSP_STATUS (*FnLinkHandshakeStart) (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

Post Conditions

See Also

11.11 FnLinkHandshakeComplete

Signature of function that completes the handshake process toward specified DSP on the Link Driver.

Syntax

```
typedef DSP_STATUS (*FnLinkHandshakeComplete) (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

Post Conditions

See Also

11.12 LinkInterface

Interface functions exported by the Link Driver.

Syntax

```
typedef struct LinkInterface_tag {
    FnLinkInitialize        initialize        ;
    FnLinkFinalize          finalize          ;
    FnLinkOpenChannel       openChannel       ;
    FnLinkCloseChannel      closeChannel      ;
    FnLinkCancelIO          cancelChannel     ;
    FnLinkIORequest         ioRequest         ;
    FnLinkScheduleDpc       scheduleDpc       ;
    FnLinkHandshakeSetup    handshakeSetup    ;
    FnLinkHandshakeStart    handshakeStart    ;
    FnLinkHandshakeComplete handshakeComplete ;
} LinkInterface ;
```

Fields

FnLinkInitialize	initialize Function pointer to the Link Driver initialize function.
FnLinkFinalize	finalize Function pointer to the Link Driver finalize function.
FnLinkOpenChannel	openChannel Function pointer to the Link Driver function to open channel.
FnLinkCloseChannel	closeChannel Function pointer to the Link Driver function to close channel.
FnLinkCancelIO	cancelChannel Function pointer to the Link Driver function to cancel channel.
FnLinkIORequest	ioRequest Function pointer to the Link Driver function to request IO.
FnLinkScheduleDpc	scheduleDpc Function pointer to the Link Driver function to schedule DPC.

FnLinkHandshakeSetup	handshakeSetup Function pointer to the Link Driver function to setup handshake.
FnLinkHandshakeStart	handshakeStart Function pointer to the Link Driver function to start handshake.
FnLinkHandshakeComplete	handshakeComplete Function pointer to the Link Driver function to complete handshake.

See Also

12 ldrv.h

Defines constants and interfaces to initialize and finalize sub-component LDRV.

Path

$(DSPLINK)\gpp\src\ldrv

Revision

00.08

TYPE DEFINITIONS & STRUCTURES

12.1 LDRV_Object

Structure to hold all component wide globals.

Syntax

```
typedef struct LDRV_Object_tag {
    Uint32          numDsps       ;
    Uint32          numLinkTables ;
    Uint32          numMmuTables  ;
    DspObject *     dspObjects    ;
    LinkAttrs **    linkTables    ;
    DspMmuEntry **  mmuTables     ;

#if defined (MSGQ_COMPONENT)
    Uint32          numMqas       ;
    Uint32          numMqts       ;
    Uint32          localMqt      ;
    MqaObject *     mqaObjects    ;
    MqtObject *     mqtObjects    ;
#endif

#if defined (DDSP_PROFILE)
    ProcStats       procStats     ;
#if defined (CHNL_COMPONENT)
    ChnlStats       chnlStats     ;
#endif
#if defined (MSGQ_COMPONENT)
    MsgqStats       msgqStats     ;
#endif
#endif
} LDRV_Object ;
```

Fields

Uint32	numDsps Number of DSPs connected to the GPP.
Uint32	numLinkTables Number of link tables specified in configuration database.
Uint32	numMmuTables

	Number of MMU tables specified in configuration database.
DspObject *	dspObjects Array of DSP objects.
LinkAttrs **	linkTables Array of pointers to link tables.
DspMmuEntry **	mmuTables Array of pointers to MMU tables.
Uint32	numMqas Number of allocators.
Uint32	numMqts Number of transports.
Uint32	localMqt Local transport identifier.
MqaObject *	mqaObjects Array of allocator objects.
MqtObject *	mqtObjects Array of transport objects.
ProcStats	procStats Statistics object for processor subcomponent.
ChnlStats	chnlStats Statistics object for channel subcomponent.
MsgqStats	msgqStats Statistics object for messaging subcomponent.

See Also

FUNCTIONS

12.2 LDRV_Initialize

Initializes the LDRV sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

None

Post Conditions

None

See Also

LDRV_Finalize

12.3 LDRV_Finalize

Releases resources used by LDRV sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

None

Post Conditions

None

See Also

LDRV_Initialize

13 ldrv_proc.h

Provides Interface Definitions for Link Driver PROC component.

Path

`$(DSPLINK)\gpp\src\ldrv`

Revision

`00.08`

FUNCTIONS

13.1 LDRV_PROC_Initialize

Allocates resources at GPP side that are required for using DSP.

It also sets up the connection to DSP from the GPP and other associated peripheral hardware.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Initialize (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP which is to be initialized.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

dspID must be valid.

Post Conditions

DSP is put in Reset state on success.

See Also

`LDRV_PROC_Finalize`

13.2 LDRV_PROC_Finalize

Releases all the resources created on GPP side for the specified DSP.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Finalize (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP which is to be finalized.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EWRONGTATE	LDRV_PROC_Initialize wasn't called before this function.

Pre Conditions

dspId must be valid.

DSP must not be in Unknown state.

Post Conditions

DSP is put in Idle state on success.

See Also

LDRV_PROC_Initialize

13.3 LDRV_PROC_Start

Starts execution of DSP from the specified location.

This function calls LDRV_DSP_Start to actually start the processor.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Start (IN ProcessorId dspId, IN Uint32 dspAddr) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP which is to be started.	
IN	Uint32	dspAddr
	Location from where to start.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EWRONGSTATE	DSP not in the right state to execute this function.

Pre Conditions

dspId must be valid.

DSP must be in either Loaded or Stopped state.

Post Conditions

DSP is put in Started state on success.

See Also

LDRV_PROC_Write
LDRV_PROC_Stop

13.4 LDRV_PROC_Stop

Stops the execution of DSP. It does so by the call to LDRV_DSP_Stop.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Stop (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP, which is to be stopped.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EWRONGSTATE	DSP not in the right state to execute this function.

Pre Conditions

dspId must be valid.

DSP must be in either Started or Stopped state.

Post Conditions

DSP is put in Stopped state on success.

See Also

LDRV_PROC_Start

13.5 LDRV_PROC_Idle

Puts the processor in idle mode, which means that read and write can be done to the processor.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Idle (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP which is to be stopped.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EWRONGSTATE	LDRV_PROC_Initialize wasn't called before this function.

Pre Conditions

dspId must be valid.

DSP must not be in Unknown state.

Post Conditions

DSP is put in Idle state on success.

See Also

LDRV_PROC_Initialize

13.6 LDRV_PROC_Read

Reads the DSP's memory space. This function calls LDRV_DSP_Read.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Read (IN     ProcessorId  dspId,
                IN     Uint32       dspAddr,
                IN     Endianism    endianInfo,
                IN OUT Uint32 *     numBytes,
                OUT    Uint8 *      buffer) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP whose memory is to be read.	
IN	Uint32	dspAddr
	Address from where to read.	
IN	Endianism	endianInfo
	This specifies endianness of the data.	
IN OUT	Uint32 *	numBytes
	Number of bytes to read from the location.	
OUT	Uint8 *	buffer
	Buffer in which store the read data.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EWRONGSTATE	DSP not in the right state to execute this function.

Pre Conditions

dspId must be valid.

buffer must be valid.

numBytes must be valid.

DSP must not be in Unknown or Reset state.

Post Conditions

None

See Also

LDRV_PROC_Idle
LDRV_PROC_Write

13.7 LDRV_PROC_Write

Writes to DSP's memory space. This function calls LDRV_DSP_Write.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Write (IN  ProcessorId  dspId,
                 IN  Uint32       dspAddr,
                 IN  Endianism    endianInfo,
                 IN  Uint32       numBytes,
                 IN  Uint8 *      buffer) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP whose memory is to be written.	
IN	Uint32	dspAddr
	Address to which we need to write.	
IN	Endianism	endianInfo
	This specifies endianness of the data.	
IN	Uint32	numBytes
	Number of bytes to read from the location.	
IN	Uint8 *	buffer
	Buffer in which to store the data to write.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EWRONGSTATE	DSP not in the right state to execute this function.

Pre Conditions

dspId must be valid.

buffer must be valid.

DSP must not be in Unknown state.

Post Conditions

DSP is put in Loaded state on success.

See Also

LDRV_PROC_Idle
LDRV_PROC_Read

13.8 LDRV_PROC_GetState

Returns the current state of the processor.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_GetState (IN ProcessorId dspId, OUT ProcState * procState) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP which is to be stopped.	
OUT	ProcState *	procState
	Placeholder for DSP Status.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

dspId must be valid.

procStatus must be valid.

Post Conditions

None

See Also

None

13.9 LDRV_PROC_SetState

Sets the current state of processor to the specified state.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_SetState (IN ProcessorId dspId, IN ProcState procState) ;
```

Arguments

IN	ProcessorId	dspId
	DSP ID of DSP, which is to be stopped.	
IN	ProcState	procState
	State to which the processor state is to be set.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

dspId must be valid.

Post Conditions

None

See Also

None

13.10 LDRV_PROC_Control

Provides a hook to perform device dependent control operations.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Control (IN  ProcessorId dspId,
                   IN  Int32       cmd,
                   OPT Pvoid       arg) ;
```

Arguments

IN	ProcessorId	dspId
	DSP Identifier.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

Post Conditions

None

See Also

None

13.11 LDRV_PROC_Instrument

Gets the instrumentation information related to PROC's

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_PROC_Instrument (IN ProcessorId procId, OUT ProcInstrument *
retVal) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor.	
OUT	ProcInstrument *	retVal
	OUT argument to contain the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	retVal is invalid.

Pre Conditions

retVal must be a valid pointer.

Post Conditions

None

See Also

None

13.12 LDRV_PROC_Debug

Prints out debug information of PROC module.

It prints all the important information maintained by this module.

Syntax

```
NORMAL_API
Void
LDRV_PROC_Debug (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId	Identifier for processor.

Return Values

Pre Conditions

None

Post Conditions

None

See Also

None

14 ldrv_chnl.h

Link driver's channel module interface.

Path

$(DSPLINK)\gpp\src\ldrv

Revision

00.17

CONSTANTS

14.1 IO Completion State flags.

Status of completion.

Syntax

```
/* IO Completed.              */
#define LDRV_CHNL_IOCSTATE_COMPLETE     0x0000

/* IO was cancelled           */
#define LDRV_CHNL_IOCSTATE_CANCELED     0x0002

/* Wait for IOC timed out.    */
#define LDRV_CHNL_IOCSTATE_TIMEOUT      0x0008

/* End Of Stream reached.     */
#define LDRV_CHNL_IOCSTATE_EOS          0x8000
```

See Also

ENUMERATIONS

14.2 ChannelState

Channel State type.

Syntax

```
typedef enum  {
    ChannelState_Ready     = 0x01,
    ChannelState_Idled     = 0x02,
    ChannelState_EOS       = 0x04,
    ChannelState_Closed    = 0x08
} ChannelState ;
```

Enum Values

ChannelState_Ready	Indicates channel is ready.
ChannelState_Idled	Indicates channel is idled.
ChannelState_EOS	Indicates channel is in End of Stream state.
ChannelState_Closed	Indicates channel is in closed state.

See Also

14.3 IOState

Completion state of IO on a channel.

Syntax

```
typedef enum  {
    IOState_Completed    = 1,
    IOState_NotCompleted = 2
} IOState ;
```

Enum Values

IOState_Completed	Indicates completion of IO for an IO request on a channel.
IOState_NotCompleted	Indicates non-completion of IO for an IO request on a channel.

See Also

TYPE DEFINITIONS & STRUCTURES

14.4 LDRVChnlIOInfo

Signature of callback function that runs when the buffer has been received from / sent to the DSP.

LDRV Channel IO information structure.

Syntax

```
typedef DSP_STATUS (*FnLdrvChnlCallback) (ProcessorId   procId,
                                          DSP_STATUS    statusOfIo,
                                          Uint8  *      buffer,
                                          Uint32        size,
                                          Pvoid         arg) ;

typedef struct LDRVChnlIOInfo_tag {
    Pvoid               buffer           ;
    Uint32              size             ;
    Uint32              arg              ;
    IOState             completionStatus ;
    FnLdrvChnlCallback  callback         ;
    Bool                dpcContext       ;
} LDRVChnlIOInfo ;
```

Arguments

ProcessorId	procId
Processor Identifier.	
DSP_STATUS	statusOfIo
Status of the IO requested.	
Uint8 *	buffer
Pointer to the message buffer.	
Uint32	size
Size of the message buffer.	
Pvoid	arg
Argument given to the callback function.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

See Also

14.5 LDRVChnlObject

LDRV Channel object.

Syntax

```
typedef struct LDRVChnlObject_tag {
    Uint32          signature     ;
    ChannelState    chnlState     ;
    List *          freeList      ;
    List *          requestList   ;
    List *          completedList ;
    ChannelAttrs    attrs         ;
    SyncEvObject *  syncEvent     ;
    SyncEvObject *  chnlIdleSync  ;
} LDRVChnlObject ;
```

Fields

Uint32	signature Signature of object.
ChannelState	chnlState State of channel.
List *	freeList List for free chirps.
List *	requestList List for requested chirps.
List *	completedList List for completed chirps.
ChannelAttrs	attrs Attributes of this channel.
SyncEvObject *	syncEvent Event to be signaled when some IO is completed for this channel.
SyncEvObject *	chnlIdleSync Sync event used by channel idle function call.

See Also

14.6 LDRVChnlIRP

CHIRP (Channel Input/output Request Packet) data structure.

Syntax

```
typedef struct LDRVChnlIRP_tag {
    ListElement         link          ;
    Uint8  *            buffer        ;
    Uint32              arg           ;
    Uint32              size          ;
    Uint32              iocStatus     ;
    FnLdrvChnlCallback  callback      ;
} LDRVChnlIRP ;
```

Fields

ListElement	link List element header needed for this structure.
Uint8 *	buffer Buffer to be filled/emptied.
Uint32	arg Issue reclaim argument.
Uint32	size Buffer length.
Uint32	iocStatus Status of IO Completion.
FnLdrvChnlCallback	callback Optional callback function.

See Also

FUNCTIONS

14.7 LDRV_CHNL_Initialize

Create and initializes resources needed for channel objects (maintained internally).

These channels can be opened later on by call to LDRV_CHNL_Open.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Initialize (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

Post Conditions

None.

See Also

LDRV_CHNL_Finalize

14.8 LDRV_CHNL_Finalize

Deletes all channel objects created for this DSP.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Finalize (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

Post Conditions

None.

See Also

LDRV_CHNL_Initialize

14.9 LDRV_CHNL_Open

Opens and prepares a channel for use.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Open (IN ProcessorId      procId,
                IN ChannelId        chnlId,
                IN ChannelAttrs *   attrs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	ChannelAttrs *	attrs
	Channel attributes.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EINVALIDARG	Invaild arguments.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

chnlId must be valid.

attrs must be a valid pointer.

Post Conditions

Channel must be in Idled state in case of success.

See Also

LDRV_CHNL_Close

14.10 LDRV_CHNL_Close

Closes the channel. No I/O can be performed on a closed channel.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Close (IN ProcessorId procId,
                 IN ChannelId   chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EINVALIDARG	Invaild arguments.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None.

See Also

LDRV_CHNL_Open

14.11 LDRV_CHNL_AddIORequest

Adds an IO request to a channel. An IO request may be a request for transferring a buffer from GPP side to DSP side or vice-versa.

The direction of data transfer is decided by the mode of channel.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_AddIORequest (IN ProcessorId      procId,
                        IN ChannelId        chnlId,
                        IN LDRVChnlIOInfo * ioInfo) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	LDRVChnlIOInfo *	ioInfo
	Information of IO to be performed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EINVALIDARG	Invaild arguments.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

chnlId must be valid.

ioInfo must be a valid pointer.

Post Conditions

None.

See Also

LDRV_CHNL_GetIOCompletion

14.12 LDRV_CHNL_GetIOCompletion

Waits for a specified amount of time for an I/O completion event on a channel.

Upon successful completion, a buffer is returned as part of ioInfo structure. A filled buffer is returned in case of an 'Input' channel and an empty buffer is returned in case of an 'Output' channel.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_GetIOCompletion (IN  ProcessorId      procId,
                           IN  ChannelId        chnlId,
                           IN  Uint32           timeout,
                           OUT LDRVChnlIOInfo * ioInfo) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Uint32	timeout
	Timeout for waiting.	
OUT	LDRVChnlIOInfo *	ioInfo
	Information of completed request.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EINVALIDARG	Invaild arguments.
DSP_EFAIL	General failure.
CHNL_E_NOIOC	Timeout parameter was "NO_WAIT", yet no I/O completions were queued.

Pre Conditions

procId must be valid.

chnlId must be valid.

ioInfo must be a valid pointer.

Post Conditions

Buffer returned as part of ioInfo is non-NULL in case of Successful completion, otherwise it is NULL.

See Also

`LDRV_CHNL_AddIORequest`

14.13 LDRV_CHNL_AddIOCompletion

Notification for the completion of IO.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_AddIOCompletion (IN ProcessorId   procId,
                           IN ChannelId     chnlId,
                           IN LDRVChnlIRP * chirp) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	LDRVChnlIRP *	chirp
	CHIRP on which completion is done.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invaild arguments.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

chnlId must be valid.

chirp must be valid.

Post Conditions

None.

See Also

None.

14.14 LDRV_CHNL_Idle

In case of input mode channel this function discards all pending input requests from the channel. In case of output mode channel, action of this function depends upon the flush parameter and is as follows: If flush is TRUE this function will block till all output buffers are transferred to the DSP.

If flush is FALSE this function will discard all the output requests pending on this channel without blocking.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Idle (IN ProcessorId  procId,
                IN ChannelId    chnlId,
                IN Bool         flush) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Bool	flush
	Boolean parameter tells whether to block or not for output mode channels.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EINVALIDARG	Invaild arguments.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

chnlId must be valid.

flush argument must be valid.

Post Conditions

IO request list is empty in case of successful completion.

See Also

None.

14.15 LDRV_CHNL_Control

Provides a hook to perform device dependent control operations on channels.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Control (IN  ProcessorId    procId,
                   IN  ChannelId      chnlId,
                   IN  Int32          cmd,
                   OPT Pvoid          arg) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None

See Also

PMGR_CHNL_Initialize

14.16 LDRV_CHNL_GetChannelMode

Gets the channel mode.

Syntax

```
NORMAL_API
ChannelMode
LDRV_CHNL_GetChannelMode (IN ProcessorId   procId,
                          IN ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

mode	Channel mode.

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None.

See Also

None.

14.17 LDRV_CHNL_GetChannelState

Gets the channel state.

Syntax

```
NORMAL_API
ChannelState
LDRV_CHNL_GetChannelState (IN ProcessorId   procId,
                           IN ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

Channel state.

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None.

See Also

None.

14.18 LDRV_CHNL_SetChannelState

Sets the channel state.

Syntax

```
NORMAL_API
Void
LDRV_CHNL_SetChannelState (IN ProcessorId   procId,
                           IN ChannelId     chnlId,
                           IN ChannelState  state) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	
IN	ChannelState	state
	State of the channel to be set.	

Return Values

None.

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None.

See Also

None.

14.19 LDRV_CHNL_GetChannelEndianism

Gets channel endianism information.

Syntax

```
NORMAL_API
Endianism
LDRV_CHNL_GetChannelEndianism (IN ProcessorId   procId,
                               IN ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

endianism information

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None.

See Also

None.

14.20 LDRV_CHNL_ChannelHasMoreChirps

Returns TRUE if the channel has more chirps in the IO request queue.

Syntax

```
NORMAL_API
Bool
LDRV_CHNL_ChannelHasMoreChirps (IN ProcessorId   procId,
                                IN ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

TRUE	Channel has more chirps in IO request queue.
FALSE	Channel does not have more chirps available in the queue.

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None.

See Also

None.

14.21 LDRV_CHNL_GetRequestChirp

Gets a chirp from request queue of a channel.

Syntax

```
NORMAL_API
LDRVChnlIRP *
LDRV_CHNL_GetRequestChirp (IN ProcessorId   procId,
                           IN ChannelId     chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

CHIRP from requested queue.

Pre Conditions

procId must be valid.

chnlId must be valid.

Post Conditions

None.

See Also

None.

14.22 LDRV_CHNL_HandshakeSetup

It does initializations related to handshake procedure that are required before starting of DSP.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_HandshakeSetup (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

dspId must be valid.

Post Conditions

None.

See Also

None.

14.23 LDRV_CHNL_Handshake

Does the necessary handshake (if required) between for the channels across GPP & DSP.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Handshake (IN ProcessorId procId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

None.

Post Conditions

None.

See Also

None.

14.24 LDRV_CHNL_Instrument

Gets the instrumentation information related to the specified channel.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_CHNL_Instrument (IN  ProcessorId       procId,
                      IN  ChannelId         chnlId,
                      OUT ChnlInstrument *  retVal) ;
```

Arguments

IN	ProcessorId	procId
	Identifier for processor.	
IN	ChannelId	chnlId
	Identifier for channel for which instrumentation information is to be obtained.	
OUT	ChnlInstrument *	retVal
	OUT argument to contain the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	retVal is invalid.

Pre Conditions

procId must be valid.

chirp must be valid.

retVal must be a valid pointer.

Post Conditions

None

See Also

None

14.25 LDRV_CHNL_Debug

Prints out debug information of CHNL module. It will print all the important data structures and variable of this module.

Syntax

```
NORMAL_API
Void
LDRV_CHNL_Debug (ProcessorId procId, ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

None.

Pre Conditions

procId must be valid.

chirp must be valid.

Post Conditions

None.

See Also

None.

15 ldrv_msgq.h

Defines the interface and structures of LDRV MSGQ driver.

Path

$(DSPLINK)\gpp\src\ldrv

Revision

00.05

CONSTANTS

15.1 ID_MSGCHNL_TO_DSP

This macro defines the ID of the messaging channel to the DSP.

Syntax

```
#define ID_MSGCHNL_TO_DSP MAX_CHANNELS
```

See Also

15.2 ID_MSGCHNL_FM_DSP

This macro defines the ID of the messaging channel from the DSP.

Syntax

```
#define ID_MSGCHNL_FM_DSP (MAX_CHANNELS + 1)
```

See Also

15.3 DSPLINK_DSPMSGQ_NAME

This macro defines the prefix to the names of all MSGQs created on the DSP for communication with the GPP.

Syntax

```
#define DSPLINK_DSPMSGQ_NAME    "DSPLINK_DSP00MSGQ"
```

See Also

15.4 DSPLINK_GPPMSGQ_NAME

This macro defines the prefix to the names of all MSGQs created on the GPP for communication with the DSP.

Syntax

```
#define DSPLINK_GPPMSGQ_NAME    "DSPLINK_GPPMSGQ"
```

See Also

ENUMERATIONS

15.5 LdrvMsgqStatus

This enumeration defines the possible states of the MSGQ object.

Syntax

```
typedef enum {
    LdrvMsgqStatus_Empty         = 0,
    LdrvMsgqStatus_Inuse         = 1,
    LdrvMsgqStatus_LocatePending = 2
} LdrvMsgqStatus ;
```

Enum Values

LdrvMsgqStatus_Empty	The message queue is empty.
LdrvMsgqStatus_Inuse	The message queue is in use.
LdrvMsgqStatus_LocatePending	The message queue is waiting for the completion of locate.

See Also

TYPE DEFINITIONS & STRUCTURES

15.6 LdrvMsgqObject

Forward declaration of LDRV MSGQ object .

Forward declaration of the handle to LDRV MSGQ object .

This structure defines the MSGQ object. It includes all information specific to a particular MSGQ.

Syntax

```
typedef struct LdrvMsgqObject_tag LdrvMsgqObject ;

typedef LdrvMsgqObject * LdrvMsgqHandle ;


struct LdrvMsgqObject_tag {
    Uint16           msgqId ;
    SyncSemObject *  getSem ;
    Pvoid            mqtRepository ;
    FnMqtGet         mqtGet ;
    FnMqtPut         mqtPut ;
    LdrvMsgqStatus   msgqStatus ;
} ;
```

Fields

Uint16	msgqId ID of the MSGQ.
SyncSemObject *	getSem Pointer to the semaphore to be used for waiting for messages on this MSGQ.
Pvoid	mqtRepository Handle to the MQT instance for this MSGQ. This object is specific to each MQT, and contains all information needed by it for interaction with this MSGQ. There is one instance of this object for every MSGQ.
FnMqtGet	mqtGet Pointer to the mqtGet () function of the transport. This pointer is replicated within this structure for faster access in time critical MSGQ_Get () function.
FnMqtPut	mqtPut Pointer to the mqtPut () function of the transport. This pointer is replicated within this structure for faster access

	in time critical MSGQ_Put () function.
LdrvMsgqStatus	msgqStatus State of the MSGQ.

See Also

15.7 LdrvMsgqState

This structure defines the MSGQ state object. It includes all global information required by the MSGQ component.

Syntax

```
typedef struct LdrvMsgqState_tag {
    LdrvMsgqAllocatorObj * allocators ;
    LdrvMsgqTransportObj * transports ;
    Uint16                 numAllocators ;
    Uint16                 numTransports ;
    Uint16                 localTransportId ;
    Uint16                 mqtMap [MAX_PROCESSORS] ;
    MsgQueueId             errorHandlerMsgq ;
    AllocatorId            errorMqaId ;
} LdrvMsgqState ;
```

Fields

LdrvMsgqAllocatorObj *	allocators Array of allocator objects.
LdrvMsgqTransportObj *	transports Array of transport objects, one for every processor in the system. This includes the local processor, as well as any other processors to which the local processor is connected.
Uint16	numAllocators Number of allocators configured in the system.
Uint16	numTransports Number of transports configured in the system.
Uint16	localTransportId ID of the local transport. The local MQT ID is used to index into the MQT objects table whenever the local MQT needs to be accessed.
Uint16	mqtMap Mapping of the processor ID to the MQT ID. This information is obtained through the CFG, and used for converting the processor ID into the MQT ID during MSGQ API calls.
MsgQueueId	errorHandlerMsgq

	ID of the MSGQ registered by the user as an error handler. If no error handler MSGQ has been registered by the user, the value of this field is MSGQ_INVALID_ID.
`AllocatorId`	`errorMqaId` ID of the allocator to be used for allocating the asynchronous error messages, if the user has registered an error handler MSGQ. If no error handler MSGQ has been registered by the user, the value of this field is MSGQ_INVALID_ID.

See Also

FUNCTIONS

15.8 LDRV_MSGQ_Setup

This function initializes the MSGQ component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Setup () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must not be initialized.

Post Conditions

Component must be initialized upon successful completion otherwise it must be uninitialized.

See Also

LDRV_MSGQ_Destroy

15.9 LDRV_MSGQ_Destroy

This function finalizes the MSGQ component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Destroy () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

Post Conditions

Component must be uninitialized.

See Also

LDRV_MSGQ_Setup

15.10 LDRV_MSGQ_AllocatorOpen

This function opens the MQA component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_AllocatorOpen (IN  AllocatorId mqaId,
                         IN  Pvoid       mqaAttrs,
                         OUT Pvoid *     mqaInfo) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be opened.	
IN	Pvoid	mqaAttrs
	Attributes for initialization of the MQA component.	
OUT	Pvoid *	mqaInfo
	Location to receive the handle to the initialized MQA state object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

mqaAttrs must be valid.

Post Conditions

None

See Also

LDRV_MsgqAllocatorAttrs

15.11 LDRV_MSGQ_AllocatorClose

This function closes the MQA component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_AllocatorClose (IN  AllocatorId mqaId) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be closed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

Post Conditions

None

See Also

LDRV_MSGQ_AllocatorOpen

15.12 LDRV_MSGQ_TransportOpen

This function initializes the MQT component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_TransportOpen (IN  TransportId mqtId, IN  Pvoid mqtAttrs) ;
```

Arguments

IN	TransportId	mqtId
	ID of the MQT to be opened.	
IN	Pvoid	mqtAttrs
	Attributes for initialization of the MQT component. The structure of the expected attributes is specific to every MQT.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

mqtAttrs must be valid.

Post Conditions

None

See Also

LDRV_MsgqTransportAttrs

15.13 LDRV_MSGQ_TransportClose

This function finalizes the MQT component.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_TransportClose (IN  TransportId mqtId) ;
```

Arguments

IN	TransportId	mqtId
	ID of the MQT to be closed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

Post Conditions

None

See Also

LDRV_MSGQ_AllocatorOpen

15.14 LDRV_MSGQ_Create

This function creates the message queue to be used for receiving messages, identified through the specified MSGQ ID.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Create (IN MsgQueueId  msgqId, IN OPT MsgqAttrs * msgqAttrs)
;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the message queue to be created.	
IN OPT	MsgqAttrs *	msgqAttrs
	Optional attributes for creation of the MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

Post Conditions

None

See Also

None.

15.15 LDRV_MSGQ_Delete

This function deletes the message queue identified by the specified MSGQ ID.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Delete (IN  MsgQueueId msgqId) ;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the message queue to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

Post Conditions

None

See Also

None.

15.16 LDRV_MSGQ_Locate

This function verifies the existence and status of the message queue identified by the specified MSGQ ID, on the specified processor.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Locate (IN  ProcessorId       procId,
                  IN  MsgQueueId        msgqId,
                  IN  MsgqLocateAttrs * attrs) ;
```

Arguments

IN	ProcessorId	procId
	ID of the processor on which the MSGQ is to be located.	
IN	MsgQueueId	msgqId
	ID of the message queue to be located.	
IN	MsgqLocateAttrs *	attrs
	Attributes for location of the MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_ENOTFOUND	The message queue does not exist on the specified processor.
DSP_ETIMEOUT	Timeout occurred while locating the MSGQ.
DSP_EMEMORY	Operation failed due to memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

procId must be valid.

attrs must not be NULL.

Post Conditions

None

See Also

None.

15.17 LDRV_MSGQ_Release

This function releases the MSGQ located through an earlier call.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Release (IN  ProcessorId procId, IN  MsgQueueId  msgqId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	MsgQueueId	msgqId
	ID of the message queue to be released.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

procId must be valid.

Post Conditions

None

See Also

None.

15.18 LDRV_MSGQ_Alloc

This function allocates a message, and returns the pointer to the user.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Alloc (IN  AllocatorId mqaId, IN  Uint16 size, OUT MsgqMsg *
msg) ;
```

Arguments

IN	AllocatorId	mqaId
	ID of the MQA to be used for allocating this message.	
IN	Uint16	size
	Size of the message to be allocated.	
OUT	MsgqMsg *	msg
	Location to receive the allocated message.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

msg must be valid.

Post Conditions

None

See Also

None.

15.19 LDRV_MSGQ_Free

This function frees a message.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Free (IN  MsgqMsg msg) ;
```

Arguments

IN	MsgqMsg	msg
	Pointer to the message to be freed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

msg must be valid.

Post Conditions

None

See Also

None.

15.20 LDRV_MSGQ_Put

This function sends a message to the specified MSGQ on the specified processor.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Put (IN     ProcessorId procId,
               IN     MsgQueueId  destMsgqId,
               IN     MsgqMsg     msg,
               IN OPT Uint16      msgId,
               IN OPT MsgQueueId  srcMsgqId) ;
```

Arguments

IN	ProcessorId	procId
	ID of the processor on which the MSGQ to which the message is to be sent, exists.	
IN	MsgQueueId	destMsgqId
	ID of the destination MSGQ.	
IN	MsgqMsg	msg
	Pointer to the message to be sent to the destination MSGQ.	
IN OPT	Uint16	msgId
	Optional message ID to be associated with the message.	
IN OPT	MsgQueueId	srcMsgqId
	ID of the source MSGQ to receive reply messages (if any).	

Return Values

DSP_SOK	Operation successfully completed.
DSP_ENOTFOUND	The message queue does not exist. This implies that the MSGQ has not been located before this call was made.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

msg must be valid.

Post Conditions

None

See Also

None.

15.21 LDRV_MSGQ_Get

This function receives a message on the specified MSGQ.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Get (IN  MsgQueueId msgqId, IN  Uint32 timeout, OUT MsgqMsg *
msg) ;
```

Arguments

IN	MsgQueueId	msgqId
	ID of the MSGQ on which the message is to be received.	
IN	Uint32	timeout
	Timeout value to wait for the message (in milliseconds).	
OUT	MsgqMsg *	msg
	Location to receive the message.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_ETIMEOUT	Timeout occurred while receiving the message.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

msg must be valid.

Post Conditions

None

See Also

None.

15.22 LDRV_MSGQ_GetReplyId

This function extracts the MSGQ ID and processor ID to be used for replying to a received message.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_GetReplyId (IN  MsgqMsg       msg,
                      OUT ProcessorId * procId,
                      OUT MsgQueueId *  msgqId) ;
```

Arguments

IN	MsgqMsg	msg
	Message, whose reply MSGQ ID is to be extracted.	
OUT	ProcessorId *	procId
	Location to retrieve the ID of the processor where the reply MSGQ resides.	
OUT	MsgQueueId *	msgqId
	Location to retrieve the ID of the reply MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

msg must be valid.

procId must be valid.

msgqId must be valid.

Post Conditions

None

See Also

None

15.23 LDRV_MSGQ_SetErrorHandler

This function allows the user to designate a MSGQ as an error-handler MSGQ to receive asynchronous error messages from the transports.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_SetErrorHandler (IN  MsgQueueId msgqId, IN  Uint16 mqaId) ;
```

Arguments

IN	MsgQueueId	msgqId
	Message queue to receive the error messages.	
IN	Uint16	mqaId
	ID indicating the allocator to be used for allocating the error messages.	

Return Values

DSP_SOK	The error handler has been successfully set.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

msgqId must be valid.

mqaId must be valid.

Post Conditions

None

See Also

None

15.24 LDRV_MSGQ_SendErrorMsg

This function sends an asynchronous error message of a particular type to the user-defined error handler MSGQ.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_SendErrorMsg (IN  MsgqErrorType errorType,
                        IN  Pvoid         arg1,
                        IN  Pvoid         arg2,
                        IN  Pvoid         arg3) ;
```

Arguments

IN	MsgqErrorType	errorType
	Type of the error.	
IN	Pvoid	arg1
	First argument dependent on the error type.	
IN	Pvoid	arg2
	Second argument dependent on the error type.	
IN	Pvoid	arg3
	Third argument dependent on the error type.	

Return Values

DSP_SOK	The error message has been successfully sent.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

Post Conditions

None

See Also

None

15.25 LDRV_MSGQ_NotImpl

This function should be used in interface tables where some functions are not being implemented.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_NotImpl () ;
```

Arguments

None

Return Values

DSP_ENOTIMPL	This function is not implemented.

Pre Conditions

None

Post Conditions

None

See Also

None

15.26 LDRV_MSGQ_Instrument

This function gets the instrumentation information related to the specified message queue.

Syntax

```
EXPORT_API
DSP_STATUS
LDRV_MSGQ_Instrument (IN  ProcessorId      procId,
                      IN  MsgQueueId       msgqId,
                      OUT MsgqInstrument * retVal) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	MsgQueueId	msgqId
	Message queue identifier.	
OUT	MsgqInstrument *	retVal
	Location to retrieve the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

The component must be initialized.

retVal must be valid.

Post Conditions

None

See Also

MsgqInstrument

15.27 LDRV_MSGQ_Debug

This function prints the current status of the MSGQ subcomponent.

Syntax

```
EXPORT_API
Void
LDRV_MSGQ_Debug (IN  ProcessorId procId, IN  MsgQueueId msgqId) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	MsgQueueId	msgqId
	Message queue identifier.	

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

16 ldrv_io.h

Defines interfaces exposed by LDRV_IO subcomponent. These services are used by LDRV_CHNL for performing IO operations. It uses services from (LDRV_)DSP subcomponent for carrying out the tasks.

Path

$(DSPLINK)\gpp\src\ldrv

Revision

00.09

FUNCTIONS

16.1 LDRV_IO_Initialize

This function allocates and initializes resources used by this component and registers interrupt handler for handling data transfer interrupts from DSP.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_Initialize (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	

Return Values

DSP_SOK	Operation completed successfully.
DSP_EMEMORY	Out of memory

Pre Conditions

dspId shall be valid.

Post Conditions

None.

See Also

LDRV_IO_Finalize
IO_ISR
LDRV_IO_DPC

16.2 LDRV_IO_Finalize

This function finalizes the I/O module for a particular DSP.

Finalizing means No further services will be provided by this module for this particular DSP.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_Finalize (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	

Return Values

DSP_SOK	Operation completed successfully.
DSP_EMEMORY	Out of memory

Pre Conditions

dspId shall be valid.

Post Conditions

None.

See Also

LDRV_IO_Initialize

16.3 LDRV_IO_OpenChannel

Open a channel for input/output.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_OpenChannel (IN ProcessorId dspId, IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	Could not open the channel successfully.

Pre Conditions

dspId shall be valid.

chnlId shall be valid.

Post Conditions

None.

See Also

None.

16.4 LDRV_IO_CloseChannel

Close a channel.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_CloseChannel (IN ProcessorId dspId, IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	Could not close the channel successfully.

Pre Conditions

dspId shall be valid.

chnlId shall be valid.

Post Conditions

None.

See Also

None.

16.5 LDRV_IO_Cancel

Cancel a channel.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_Cancel (IN ProcessorId dspId, IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	Could not close the channel successfully.

Pre Conditions

dspId shall be valid.

chnlId shall be valid.

Post Conditions

None.

See Also

None.

16.6 LDRV_IO_Request

This function sends an IO request on specified channel to the link driver.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_Request (IN ProcessorId dspId, IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	chnlId is invalid.

Pre Conditions

dspId shall be valid.

chnlId shall be valid.

This sub-component must have been initialized before calling this function.

Post Conditions

None.

See Also

LDRV_IO_Dispatch

16.7 LDRV_IO_ScheduleDPC

Schedules DPC for IO with the DSP specified by dspId.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_ScheduleDPC (IN ProcessorId dspId, IN ChannelId chnlId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	
IN	ChannelId	chnlId
	Channel Identifier.	

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

This sub-component must have been initialized before calling this function.

dspId shall be valid.

Post Conditions

None.

See Also

None.

16.8 LDRV_IO_HandshakeSetup

Does necessary initializations for handshake procedure.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_HandshakeSetup (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	Operation failed.

Pre Conditions

dspId shall be valid.

Post Conditions

None.

See Also

None.

16.9 LDRV_IO_Handshake

Does the necessary handshake (if required) between for the links across GPP & DSP.

Syntax

```
NORMAL_API
DSP_STATUS
LDRV_IO_Handshake (IN ProcessorId dspId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	

Return Values

None.

Pre Conditions

dspId shall be valid.

Post Conditions

None.

See Also

None.

16.10 LDRV_IO_Debug

Prints the current status of this subcomponent.

Syntax

```
NORMAL_API
Void
LDRV_IO_Debug (IN  ProcessorId  dspId) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	

Return Values

None.

Pre Conditions

dspId shall be valid.

Post Conditions

None.

See Also

None.

17 ldrv_mqa.h

Defines the interface and structures of LDRV MQT.

Path

`$(DSPLINK)\gpp\src\ldrv`

Revision

`01.10`

FUNCTIONS

17.1 FnMqaInitialize

Signature of the function that performs global initialization of the buffer MQA.

Syntax

```
typedef Void (*FnMqaInitialize) () ;
```

Arguments

None

Return Values

None

Pre Conditions

Post Conditions

See Also

17.2 FnMqaFinalize

Signature of the function that performs global finalization of the buffer MQA.

Syntax

```
typedef Void (*FnMqaInitialize) () ;

typedef Void (*FnMqaFinalize) () ;
```

Arguments

None

Return Values

None

Pre Conditions

Post Conditions

See Also

17.3 FnMqaOpen

Signature of the function that opens the buffer MQA and configures it according to the user attributes.

Syntax

```
typedef DSP_STATUS (*FnMqaOpen) (LdrvMsgqAllocatorHandle  mqaHandle,
                                 Pvoid                    mqaAttrs) ;
```

Arguments

`LdrvMsgqAllocatorHandle`	`mqaHandle`
Handle to the MSGQ allocator object.	
`Pvoid`	`mqaAttrs`
Attributes for initialization of the MQA component.	

Return Values

`DSP_SOK`	This component has been successfully opened.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EINVALIDARG`	Invalid argument.
`DSP_EFAIL`	General failure.

Pre Conditions

Post Conditions

See Also

17.4 FnMqaClose

Signature of the function that closes the buffer MQA.

Syntax

```
typedef DSP_STATUS (*FnMqaClose) (LdrvMsgqAllocatorHandle  mqaHandle) ;
```

Arguments

LdrvMsgqAllocatorHandle mqaHandle	
Handle to the MSGQ allocator object.	

Return Values

DSP_SOK	This component has been successfully closed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

17.5 FnMqaAlloc

Signature of the function that allocates a message buffer of the specified size.

Syntax

```
typedef DSP_STATUS (*FnMqaAlloc) (LdrvMsgqAllocatorHandle mqaHandle,
                                  Uint16 *                size,
                                  MsgqMsg *               addr) ;
```

Arguments

`LdrvMsgqAllocatorHandle mqaHandle`	
Handle to the MSGQ allocator object.	
`Uint16 * size`	
Pointer to the size of the message to be allocated.	
`MsgqMsg * addr`	
Location to receive the allocated message.	

Return Values

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EINVALIDARG`	Invalid argument.
`DSP_EFAIL`	General failure.

Pre Conditions

Post Conditions

See Also

17.6 FnMqaFree

Signature of the function that frees the message of the specified size.

Syntax

```
typedef DSP_STATUS (*FnMqaFree) (LdrvMsgqAllocatorHandle mqaHandle,
                                 MsgqMsg                 addr,
                                 Uint16                  size) ;
```

Arguments

LdrvMsgqAllocatorHandle	mqaHandle
Handle to the MSGQ allocator object.	
MsgqMsg	addr
Address of the message to be freed.	
Uint16	size
Size of the message to be freed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

TYPE DEFINITIONS & STRUCTURES

17.7 MqaInterface

This structure defines the function pointer table that must be implemented for every MQA in the system.

Syntax

```
struct MqaInterface_tag {
    FnMqaInitialize mqaInitialize ;
    FnMqaFinalize   mqaFinalize   ;
    FnMqaOpen       mqaOpen       ;
    FnMqaClose      mqaClose      ;
    FnMqaAlloc      mqaAlloc      ;
    FnMqaFree       mqaFree       ;
} ;
```

Fields

FnMqaInitialize	mqaInitialize Pointer to MQA initialization function.
FnMqaFinalize	mqaFinalize Pointer to MQA finalization function.
FnMqaOpen	mqaOpen Pointer to MQA open function.
FnMqaClose	mqaClose Pointer to MQA close function.
FnMqaAlloc	mqaAlloc Pointer to MQA function for allocating a message.
FnMqaFree	mqaFree Pointer to MQA function for freeing a message.

See Also

17.8 LdrvMsgqAllocatorObj_tag

This structure defines the allocator object. There is one instance of the allocator object per MQA in the system.

Syntax

```
struct LdrvMsgqAllocatorObj_tag {
#if defined (DDSP_DEBUG)
    Char8                   mqaName [DSP_MAX_STRLEN] ;
#endif
    MqaInterface *          mqaInterface ;
    Pvoid                   mqaInfo ;
    Uint16                  mqaId   ;
} ;
```

Fields

Char8	mqaName Name of the MQA. Used for debugging purposes only.
MqaInterface *	mqaInterface Pointer to the function table of the MQA represented by the allocator object.
Pvoid	mqaInfo State information needed by the allocator. The contents of this are allocator-specific.
Uint16	mqaId ID of the MQA represented by the allocator object.

See Also

17.9 MqaObject

This structure defines the MQA object stored in the LDRV object.

Syntax

```
struct MqaObject_tag {
    Char8           mqaName    [DSP_MAX_STRLEN] ;
    MqaInterface *  interface                   ;
} ;
```

Fields

Char8	mqaName Name of the MQA. For debugging purposes only.
MqaInterface *	interface Function pointer interface to access the functions for this MQA.

See Also

18 ldrv_mqt.h

Defines the interface and structures of LDRV MQT.

Path

`$(DSPLINK)\gpp\src\ldrv`

Revision

`01.10`

FUNCTIONS

18.1 FnMqtInitialize

Signature of the function that performs MQT initialization.

Syntax

```
typedef Void (*FnMqtInitialize) () ;
```

Arguments

None

Return Values

None

Pre Conditions

Post Conditions

See Also

18.2 FnMqtFinalize

Signature of the function that performs MQT finalization.

Syntax

```
typedef Void (*FnMqtFinalize) () ;
```

Arguments

None

Return Values

None

Pre Conditions

Post Conditions

See Also

18.3 FnMqtOpen

Signature of the function that opens the MQT and configures it according to the user attributes.

Syntax

```
typedef DSP_STATUS (*FnMqtOpen) (LdrvMsgqTransportHandle mqtHandle,
                                 Pvoid                   mqtAttrs) ;
```

Arguments

LdrvMsgqTransportHandle	mqtHandle
This is the handle to LDRV MQT transport object.	
Pvoid	mqtAttrs
Attributes for initialization of the MQT component.	

Return Values

DSP_SOK	This component has been successfully opened.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

18.4 FnMqtClose

Signature of the function that closes the MQT, and cleans up its state object.

Syntax

```
typedef DSP_STATUS (*FnMqtClose) (LdrvMsgqTransportHandle mqtHandle) ;
```

Arguments

LdrvMsgqTransportHandle mqtHandle	
This is the handle to LDRV MQT transport object.	

Return Values

DSP_SOK	This component has been successfully closed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

18.5 FnMqtCreate

Signature of the function that creates the message queue identified by the specified MSGQ ID.

Syntax

```
typedef DSP_STATUS (*FnMqtCreate) (LdrvMsgqTransportHandle mqtHandle,
                                   MsgQueueId              msgqId,
                                   MsgqAttrs *             attrs) ;
```

Arguments

`LdrvMsgqTransportHandle`	`mqtHandle`
This is the handle to LDRV MQT transport object.	
`MsgQueueId`	`msgqId`
ID of the message queue to be created.	
`MsgqAttrs *`	`attrs`
Optional attributes for creation of the MSGQ.	

Return Values

`DSP_SOK`	Operation successfully completed.
`DSP_EMEMORY`	Operation failed due to a memory error.
`DSP_EINVALIDARG`	Invalid argument.
`DSP_EFAIL`	General failure.

Pre Conditions

Post Conditions

See Also

18.6 FnMqtLocate

Signature of the function that verifies the existence and status of the message queue identified by the specified MSGQ ID.

Syntax

```
typedef DSP_STATUS (*FnMqtLocate) (LdrvMsgqTransportHandle mqtHandle,
                                   MsgQueueId              msgqId,
                                   MsgqLocateAttrs *       attrs) ;
```

Arguments

`LdrvMsgqTransportHandle`	`mqtHandle`
This is the handle to LDRV MQT transport object.	
`MsgQueueId`	`msgqId`
ID of the message queue to be located.	
`MsgqLocateAttrs *`	`attrs`
Attributes for location of the MSGQ.	

Return Values

`DSP_SOK`	Operation successfully completed.
`DSP_ENOTFOUND`	The message queue does not exist among the MSQs managed by this MQT.
`DSP_EINVALIDARG`	Invalid argument.
`DSP_EFAIL`	General failure.

Pre Conditions

Post Conditions

See Also

18.7 FnMqtDelete

Signature of the function that deletes the message queue identified by the specified MSGQ ID.

Syntax

```
typedef DSP_STATUS (*FnMqtDelete) (LdrvMsgqTransportHandle mqtHandle,
                                   MsgQueueId              msgqId) ;
```

Arguments

LdrvMsgqTransportHandle mqtHandle	
This is the handle to LDRV MQT transport object.	
MsgQueueId msgqId	
ID of the message queue to be deleted.	

Return Values

DSP_SOK	The message queue has been successfully deleted.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

18.8 FnMqtRelease

Signature of the function that releases the MSGQ located through an earlier locate / getreplyid call.

Syntax

```
typedef DSP_STATUS (*FnMqtRelease) (LdrvMsgqTransportHandle mqtHandle,
                                    MsgQueueId              msgqId) ;
```

Arguments

LdrvMsgqTransportHandle mqtHandle	
This is the handle to LDRV MQT transport object.	
MsgQueueId msgqId	
ID of the message queue to be released.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

18.9 FnMqtGet

Signature of the function that receives a message on the specified MSGQ.

Syntax

```
typedef DSP_STATUS (*FnMqtGet) (LdrvMsgqTransportHandle mqtHandle,
                                MsgQueueId              msgqId,
                                Uint32                  timeout,
                                MsgqMsg *               msg) ;
```

Arguments

LdrvMsgqTransportHandle	mqtHandle
This is the handle to LDRV MQT transport object.	
MsgQueueId	msgqId
ID of the MSGQ on which the message is to be received.	
Uint32	timeout
Timeout value to wait for the message (in milliseconds).	
MsgqMsg *	msg
Location to receive the message.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_ETIMEOUT	Timeout occurred while receiving the message.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

18.10 FnMqtPut

Signature of the function that sends a message on the specified MSGQ ID.

Syntax

```
typedef DSP_STATUS (*FnMqtPut) (LdrvMsgqTransportHandle mqtHandle,
                                MsgQueueId              msgqId,
                                MsgqMsg                 msg) ;
```

Arguments

LdrvMsgqTransportHandle	mqtHandle
This is the handle to LDRV MQT transport object.	
MsgQueueId	msgqId
ID of the destination MSGQ.	
MsgqMsg	msg
Pointer to the message to be sent to the destination MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_ENOTFOUND	The message queue does not exist. This implies that the MSGQ has not been located before this call was made.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

18.11 FnMqtGetReplyId

Signature of the function that gets the reply MSGQ ID for a particular message.

Syntax

```
typedef DSP_STATUS (*FnMqtGetReplyId)
            (LdrvMsgqTransportHandle mqtHandle,
             MsgqMsg                 msg,
             MsgQueueId *            msgqId) ;
```

Arguments

LdrvMsgqTransportHandle	mqtHandle
This is the handle to LDRV MQT transport object.	
MsgqMsg	msg
Pointer to the message.	
MsgQueueId *	msgqId
ID of the destination MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

Post Conditions

See Also

18.12 FnMqtGetById

Signature of the MQT function that receives a message having a particular MSG ID.

Syntax

```
typedef DSP_STATUS (*FnMqtGetById) (MsgQueueId  msgqId,
                                    Uint16 *    msgIds,
                                    Uint16      numIds,
                                    Uint32      timeout,
                                    MsgqMsg *   msg) ;
```

Arguments

MsgQueueId	msgqId
Message queue identifier.	
Uint16 *	msgIds
Array containing message queue identifiers.	
Uint16	numIds
Number of message queue identifiers in the array.	
Uint32	timeout
Timeout value to wait for the message (in milliseconds).	
MsgqMsg *	msg
Location to receive the allocated message.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_ETIMEOUT	Timeout occurred while receiving the message.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

Post Conditions

See Also

18.13 FnMqtInstrument

Signature of the MQT instrumentation function.

Syntax

```
typedef DSP_STATUS (*FnMqtInstrument) (LdrvMsgqTransportHandle
mqtHandle,
                                       MsgQueueId              msgqId,
                                       MsgqInstrument *        retVal)
;
```

Arguments

`LdrvMsgqTransportHandle`	`mqtHandle`
This is the handle to LDRV MQT transport object.	
`MsgQueueId`	`msgqId`
Message queue identifier.	
`MsgqInstrument *`	`retVal`
Location to retrieve the instrumentation information.	

Return Values

`DSP_SOK`	Operation successfully completed.
`DSP_EFAIL`	General failure.

Pre Conditions

Post Conditions

See Also

18.14 FnMqtDebug

Signature of the MQT debug function.

Syntax

```
typedef Void (*FnMqtDebug) (LdrvMsgqTransportHandle mqtHandle,
                            MsgQueueId              msgqId) ;
```

Arguments

LdrvMsgqTransportHandle	mqtHandle
This is the handle to LDRV MQT transport object.	
MsgQueueId	msgqId
Message queue identifier.	

Return Values

None.

Pre Conditions

Post Conditions

See Also

TYPE DEFINITIONS & STRUCTURES

18.15 MqtInterface

This structure defines the function pointer table that must be implemented for every MQT in the system.

Syntax

```
struct MqtInterface_tag {
    FnMqtInitialize   mqtInitialize ;
    FnMqtFinalize     mqtFinalize   ;
    FnMqtOpen         mqtOpen       ;
    FnMqtClose        mqtClose      ;
    FnMqtCreate       mqtCreate     ;
    FnMqtLocate       mqtLocate     ;
    FnMqtDelete       mqtDelete     ;
    FnMqtRelease      mqtRelease    ;
    FnMqtGet          mqtGet        ;
    FnMqtPut          mqtPut        ;
    FnMqtGetReplyId   mqtGetReplyId ;
    FnMqtGetById      mqtGetById    ;
#if defined (DDSP_PROFILE)
    FnMqtInstrument   mqtInstrument ;
#endif
#if defined (DDSP_DEBUG)
    FnMqtDebug        mqtDebug      ;
#endif
} ;
```

Fields

FnMqtInitialize	mqtInitialize Pointer to MQT initialization function.
FnMqtFinalize	mqtFinalize Pointer to MQT finalization function.
FnMqtOpen	mqtOpen Pointer to MQT open function.
FnMqtClose	mqtClose Pointer to MQT close function.
FnMqtCreate	mqtCreate Pointer to MQT function for creating a MSGQ.
FnMqtLocate	mqtLocate Pointer to MQT function for locating a MSGQ.

FnMqtDelete	mqtDelete Pointer to MQT function for deleting a MSGQ.
FnMqtRelease	mqtRelease Pointer to MQT function for releasing a MSGQ.
FnMqtGet	mqtGet Pointer to MQT function for receiving a message.
FnMqtPut	mqtPut Pointer to MQT function for sending a message.
FnMqtGetReplyId	mqtGetReplyId Pointer to MQT function for getting the reply MSGQ ID for a particular message.
FnMqtGetById	mqtGetById Pointer to MQT function for receiving a message having a particular MSG ID.
FnMqtInstrument	mqtInstrument Pointer to MQT Instrumentation function.
FnMqtDebug	mqtDebug Pointer to MQT debug function.

See Also

18.16 LdrvMsgqTransportObj_tag

This structure defines the common attributes of the transport object. There is one instance of the transport object per MQT in the system.

Syntax

```
struct LdrvMsgqTransportObj_tag {
#if defined (DDSP_DEBUG)
    Char8                   mqtName [DSP_MAX_STRLEN] ;
#endif
    MqtInterface *          mqtInterface ;
    Pvoid                   mqtInfo ;
    Uint16                  mqtId ;
    ProcessorId             procId ;
} ;
```

Fields

Char8	mqtName Name of the MQT. Used for debugging purposes only.
MqtInterface *	mqtInterface Pointer to the function table of the MQT represented by the transport object.
Pvoid	mqtInfo State information needed by the transport. The contents of this are transport-specific.
Uint16	mqtId ID of the MQT represented by the transport object.
ProcessorId	procId Processor Id associated with this MQT.

See Also

18.17 MqtObject

This structure defines the MQT object stored in the LDRV object.

Syntax

```
struct MqtObject_tag {
    Char8           mqtName    [DSP_MAX_STRLEN] ;
    MqtInterface *  interface                   ;
    Uint32          linkId                      ;
} ;
```

Fields

Char8	mqtName Name of the MQT. For debugging purposes only.
MqtInterface *	interface Function pointer interface to access the functions for this MQT.
Uint32	linkId ID of the link used by this MQT.

See Also

19 mqabuf.h

Defines the BUF MQA interface.

Path

$(DSPLINK)\gpp\src\ldrv

Revision

00.05

TYPE DEFINITIONS & STRUCTURES

19.1 MqaBufObj

This structure defines the buffer object for the buffer allocator.

Syntax

```
typedef struct MqaBufObj_tag {
    Uint16      msgSize ;
    BufHandle   msgList ;
} MqaBufObj ;
```

Fields

Uint16	msgSize Size of the messages in the buffer pool.
BufHandle	msgList List of messages in the buffer pool.

See Also

19.2 MqaBufState

This structure defines the allocator state object, which exists as a single instance in the system, and represents the allocator.

Syntax

```
typedef struct MqaBufState_tag {
    Uint16          numBufPools ;
    MqaBufObj     * bufPools    ;
    Uint32          phyAddr     ;
    Uint32          virtAddr    ;
    Uint32          size        ;
} MqaBufState ;
```

Fields

Uint16	numBufPools Number of buffer pools configured in the MQA.
MqaBufObj *	bufPools Array of buffer pools for various message sizes. The array is dynamically allocated of size equal to the one specified by the user.
Uint32	phyAddr Physical address of the buffer pool allocated.
Uint32	virtAddr Virtual address of the buffer pool allocated.
Uint32	size Size of memory allocated for all buffer pools.

See Also

FUNCTIONS

19.3 MQABUF_Initialize

Performs global initialization of the buffer MQA.

Syntax

```
NORMAL_API
Void
MQABUF_Initialize () ;
```

Arguments

None

Return Values

None

Pre Conditions

The component must be unintialized.

Post Conditions

None

See Also

MQABUF_Finalize

19.4 MQABUF_Finalize

Performs global finalization of the buffer MQA.

Syntax

```
NORMAL_API
Void
MQABUF_Finalize () ;
```

Arguments

None

Return Values

None

Pre Conditions

The component must be intialized.

Post Conditions

None

See Also

MQABUF_Initialize

19.5 MQABUF_Open

Opens the buffer MQA and configures it according to the user attributes.

Syntax

```
NORMAL_API
DSP_STATUS
MQABUF_Open (IN  LdrvMsgqAllocatorHandle mqaHandle, IN  Pvoid mqaAttrs)
;
```

Arguments

IN	LdrvMsgqAllocatorHandle	mqaHandle
	Handle to the MSGQ allocator object.	
IN	Pvoid	mqaAttrs
	Attributes for initialization of the MQA component.	

Return Values

DSP_SOK	This component has been successfully opened.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqaAttrs must be valid.

mqaHandle must be valid.

The component must be intialized.

Post Conditions

None

See Also

MQABUF_Attrs
MqaBufState
MQABUF_Close
BUF_Create

19.6 MQABUF_Close

This function closes the MQA and cleans up its state object.

Syntax

```
NORMAL_API
DSP_STATUS
MQABUF_Close (IN  LdrvMsgqAllocatorHandle mqaHandle) ;
```

Arguments

IN	LdrvMsgqAllocatorHandle	mqaHandle
	Handle to the MSGQ allocator object.	

Return Values

DSP_SOK	This component has been successfully closed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqaHandle must be valid.

The component must be intialized.

Post Conditions

None

See Also

MqaBufState
MQABUF_Open
BUF_Delete

19.7 MQABUF_Alloc

This function allocates a message buffer of the specified size.

Syntax

```
NORMAL_API
DSP_STATUS
MQABUF_Alloc (IN     LdrvMsgqAllocatorHandle mqaHandle,
              IN OUT Uint16 *                size,
              OUT    MsgqMsg *               addr) ;
```

Arguments

IN	LdrvMsgqAllocatorHandle	mqaHandle
	Handle to the MSGQ allocator object.	
IN OUT	Uint16 *	size
	Size of the message to be allocated. On return, it stores the actual allocated size of the message, which, for the buffer MQA is the same as the requested size on success, or zero on failure.	
OUT	MsgqMsg *	addr
	Location to receive the allocated message.	

Return Values

DSP_SOK	This component has been successfully closed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqaHandle must be valid.

size must be valid.

addr must be valid.

The component must be intialized.

Post Conditions

None

See Also

MqaBufState
MQABUF_Free

BUF_Alloc

19.8 MQABUF_Free

This function frees the message buffer given by the user of the specified size.

Syntax

```
NORMAL_API
DSP_STATUS
MQABUF_Free (IN  LdrvMsgqAllocatorHandle mqaHandle,
             IN  MsgqMsg                 addr,
             IN  Uint16                  size) ;
```

Arguments

IN	LdrvMsgqAllocatorHandle	mqaHandle
	Handle to the MSGQ allocator object.	
IN	MsgqMsg	addr
	Address of the message to be freed.	
IN	Uint16	size
	Size of the message to be freed.	

Return Values

DSP_SOK	This component has been successfully closed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqaHandle must be valid.

addr must be valid.

The component must be intialized.

Post Conditions

None

See Also

MqaBufState
MQABUF_Alloc
BUF_Free

20 lmqt.h

Defines the local MQT interface.

Path

$(DSPLINK)\gpp\src\ldrv

Revision

00.04

TYPE DEFINITIONS & STRUCTURES

20.1 LmqtObj

This structure defines the transport object, which has an instance for every MSGQ created on the processor.

Syntax

```
typedef struct LmqtObj_tag {
    List *    msgQueue ;
} LmqtObj ;
```

Fields

List *	msgQueue	Message repository to queue pending messages.

See Also

20.2 LmqtState

This structure defines the transport state object, which exists as a single instance for the local MQT.

Syntax

```
typedef struct LmqtState_tag {
    Uint16            maxNumMsgq  ;
    LdrvMsgqHandle *  msgqHandles ;
} LmqtState ;
```

Fields

Uint16	maxNumMsgq Maximum number of MSGQs that can be created on the local processor.
LdrvMsgqHandle *	msgqHandles Array of handles to the MSGQ objects for the local MSGQs.

See Also

FUNCTIONS

20.3 LMQT_Initialize

This function performs global initialization of the local MQT.

Syntax

```
NORMAL_API
Void
LMQT_Initialize () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

LDRV_MSGQ_Setup
LMQT_Finalize

20.4 LMQT_Finalize

This function performs global finalization of the local MQT.

Syntax

```
NORMAL_API
Void
LMQT_Finalize () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

LDRV_MSGQ_Destroy
LMQT_Initialize

20.5 LMQT_Open

This function opens the local MQT and configures it according to the user attributes.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Open (IN  LdrvMsgqTransportHandle mqtHandle, IN  Pvoid mqtAttrs) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	Pvoid	mqtAttrs
	Attributes required for initialization of the MQT component.	

Return Values

DSP_SOK	The local MQT has been successfully initialized.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

mqtAttrs must be valid.

Post Conditions

None

See Also

LmqtState
LmqtAttrs
LMQT_Close

20.6 LMQT_Close

This function closes the local MQT, and cleans up its state object.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Close (IN  LdrvMsgqTransportHandle mqtHandle) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	

Return Values

DSP_SOK	This component has been successfully closed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

Post Conditions

None

See Also

LmqtState
LMQT_Open

20.7 LMQT_Create

This function creates the message queue identified by the specified MSGQ ID.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Create (IN     LdrvMsgqTransportHandle mqtHandle,
             IN     MsgQueueId              msgqId,
             IN OPT MsgqAttrs *             attrs) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	ID of the message queue to be created.	
IN OPT	MsgqAttrs *	attrs
	Optional attributes for creation of the MSGQ.	

Return Values

DSP_SOK	The message queue has been successfully created.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

Post Conditions

None

See Also

LmqtObj
LMQT_Delete

20.8 LMQT_Delete

This function deletes the message queue identified by the specified MSGQ ID.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Delete (IN  LdrvMsgqTransportHandle mqtHandle, IN  MsgQueueId
msgqId) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	ID of the message queue to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

Post Conditions

None

See Also

LmqtObj
LMQT_Create

20.9 LMQT_Locate

This function verifies the existence and status of the message queue identified by the specified MSGQ ID.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Locate (IN  LdrvMsgqTransportHandle mqtHandle,
             IN  MsgQueueId              msgqId,
             IN  MsgqLocateAttrs *       attrs) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	ID of the message queue to be located.	
IN	MsgqLocateAttrs *	attrs
	Attributes for location of the MSGQ.	

Return Values

DSP_SOK	The message queue has been successfully located.
DSP_EINVALIDARG	Invalid argument.
DSP_ENOTFOUND	The message queue does not exist among the MSQs managed by this MQT.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

attrs must be valid.

Post Conditions

None

See Also

LMQT_Release

20.10 LMQT_Release

This function releases the MSGQ located through an earlier LMQT_Locate () or LMQT_GetReplyId () call.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Release (IN  LdrvMsgqTransportHandle mqtHandle, IN  MsgQueueId
msgqId) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	ID of the message queue to be released.	

Return Values

DSP_SOK	The message queue has been successfully released.
DSP_EINVALIDARG	Invalid argument.
DSP_ENOTFOUND	The message queue has not been previously located.
DSP_EFAIL	General failure

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

Post Conditions

None

See Also

LMQT_Locate

20.11 LMQT_Get

This function receives a message on the specified MSGQ.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Get (IN  LdrvMsgqTransportHandle mqtHandle,
          IN  MsgQueueId              msgqId,
          IN  Uint32                  timeout,
          OUT MsgqMsg *               msg) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	ID of the MSGQ on which the message is to be received.	
IN	Uint32	timeout
	Timeout value to wait for the message (in milliseconds).	
OUT	MsgqMsg *	msg
	Location to receive the message.	

Return Values

DSP_SOK	The message has been successfully received.
DSP_EINVALIDARG	Invalid argument.
DSP_ETIMEOUT	Timeout occurred while receiving the message.
DSP_ENOTCOMPLETE	
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

msg must be valid.

Post Conditions

None

See Also

None

20.12 LMQT_Put

This function sends a message to the specified local MSGQ.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Put (IN  LdrvMsgqTransportHandle mqtHandle,
          IN  MsgQueueId               msgqId,
          IN  MsgqMsg                  msg) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	ID of the destination MSGQ.	
IN	MsgqMsg	msg
	Pointer to the message to be sent to the destination MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_ENOTFOUND	The message queue does not exist.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

msg must be valid.

Post Conditions

None

See Also

None

20.13 LMQT_GetReplyId

This function extracts the MSGQ ID to be used for replying to a received message.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_GetReplyId (IN  LdrvMsgqTransportHandle mqtHandle,
                 IN  MsgqMsg                 msg,
                 IN  MsgQueueId *            msgqId) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgqMsg	msg
	Message, whose reply MSGQ ID is to be extracted.	
IN	MsgQueueId *	msgqId
	Location to receive the ID of the reply MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msg must be valid.

msgqId must be valid.

Post Conditions

None

See Also

None

20.14 LMQT_Instrument

This function gets the instrumentation information related to the specified message queue.

Syntax

```
NORMAL_API
DSP_STATUS
LMQT_Instrument (IN  LdrvMsgqTransportHandle mqtHandle,
                 IN  MsgQueueId              msgqId,
                 OUT MsgqInstrument *        retVal) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	Message queue identifier.	
OUT	MsgqInstrument *	retVal
	Location to retrieve the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

retVal must be valid.

Post Conditions

None

See Also

None

20.15 LMQT_Debug

This function gets the instrumentation information related to the specified message queue.

Syntax

```
NORMAL_API
Void
LMQT_Debug (IN  LdrvMsgqTransportHandle mqtHandle, IN  MsgQueueId
msgqId) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MSGQ transport object.	
IN	MsgQueueId	msgqId
	Message queue identifier.	

Return Values

None.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

Post Conditions

None

See Also

None

21 rmqt.h

Defines the remote MQT interface.

Path

$(DSPLINK)\gpp\src\ldrv

Revision

00.04

FUNCTIONS

21.1 RMQT_Initialize

This macro defines the internal ID used to identify control messages.

This enumeration defines the types of control commands that are sent between the MQTs on different processors.

This structure defines the transport object, which has an instance for every MSGQ created on the processor.

This structure defines the format of the control messages that are sent between the MQTs on different processors. This structure is common between the GPP and the DSP, and must be maintained as the same on both. To ensure this, padding must be added as required within the structure.

This structure defines the transport state object, which exists as a single instance for the remote MQT.

This function performs global initialization of the remote MQT.

Syntax

```
#define ID_RMQT_CTRL (Uint16) 0xFF00


typedef enum {
    RmqtCtrlCmd_Locate     = 0,
    RmqtCtrlCmd_LocateAck  = 1,
    RmqtCtrlCmd_Exit       = 2
} RmqtCtrlCmd ;


typedef struct RmqtObj_tag {
    SyncSemObject *    locateSem ;
} RmqtObj ;

typedef struct RmqtCtrlMsg_tag {
    MsgqMsgHeader msgHeader ;
    union {
        struct {
            Uint16      msgqId      ;
            Uint16      mqaId       ;
            Uint32      timeout     ;
            Uint32      replyHandle ;
            Uint32      arg         ;
            Uint32      semHandle   ;
```

```
            } locateMsg ;

            struct {
                Uint16      msgqId      ;
                Uint16      mqaId       ;
                Uint32      timeout     ;
                Uint32      replyHandle ;
                Uint32      arg         ;
                Uint32      semHandle   ;
                Uint16      msgqFound   ;
                Uint16      padding     ;
            } locateAckMsg ;
        } ctrlMsg ;
    } RmqtCtrlMsg ;

    typedef struct RmqtState_tag {
        LdrvMsgqTransportHandle mqtHandle    ;
        Uint16                  maxNumMsgq   ;
        Uint16                  maxMsgSize   ;
        LdrvMsgqHandle *        msgqHandles  ;
        MsgqMsg                 getBuffer    ;
        List *                  msgQueue     ;
        Uint16                  defaultMqaId ;
        ProcessorId             procId       ;
    } RmqtState ;


    NORMAL_API
    Void
    RMQT_Initialize () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

LDRV_MSGQ_Setup
RMQT_Finalize

21.2 RMQT_Finalize

This function performs global finalization of the remote MQT.

Syntax

```
NORMAL_API
Void
RMQT_Finalize () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

LDRV_MSGQ_Destroy
RMQT_Initialize

21.3 RMQT_Open

This function opens the remote MQT and configures it according to the user attributes.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_Open (IN  LdrvMsgqTransportHandle mqtHandle, IN  Pvoid mqtAttrs) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	Handle to the transport object.	
IN	Pvoid	mqtAttrs
	Attributes required for initialization of the MQT component.	

Return Values

DSP_SOK	The remote MQT has been successfully opened.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

mqtAttrs must be valid.

Post Conditions

None

See Also

RmqtState
RmqtAttrs
RMQT_Close

21.4 RMQT_Close

This function closes the remote MQT, and cleans up its state object.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_Close (IN  LdrvMsgqTransportHandle mqtHandle) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	Handle to the transport object.	

Return Values

DSP_SOK	The remote MQT has been successfully closed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

Post Conditions

None

See Also

RmqtState
RMQT_Open

21.5 RMQT_Locate

This function verifies the existence and status of the message queue identified by the specified MSGQ ID.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_Locate (IN  LdrvMsgqTransportHandle mqtHandle,
             IN  MsgQueueId              msgqId,
             IN  MsgqLocateAttrs *       attrs) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	Handle to the transport object.	
IN	MsgQueueId	msgqId
	ID of the message queue to be located.	
IN	MsgqLocateAttrs *	attrs
	Attributes for location of the MSGQ.	

Return Values

DSP_SOK	The message queue has been successfully located.
DSP_EINVALIDARG	Invalid argument.
DSP_ETIMEOUT	Timeout occurred while locating the MSGQ.
DSP_ENOTFOUND	The message queue does not exist among the MSGQs managed by this MQT.
DSP_ENOTCOMPLETE	
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

attrs must be valid.

Post Conditions

None

See Also

ID_LOCAL_PROCESSOR
RMQT_Release

21.6 RMQT_Release

This function releases the MSGQ located through an earlier RMQT_Locate () or RMQT_GetReplyId () call.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_Release (IN  LdrvMsgqTransportHandle mqtHandle, IN  MsgQueueId
msgqId) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	Handle to the transport object.	
IN	MsgQueueId	msgqId
	ID of the message queue to be released.	

Return Values

DSP_SOK	The message queue has been successfully released.
DSP_EINVALIDARG	Invalid argument.
DSP_ENOTFOUND	The message queue was not previously located.
DSP_EFAIL	General failure

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

Post Conditions

None

See Also

ID_LOCAL_PROCESSOR
RMQT_Locate

21.7 RMQT_Put

This function sends a message to the specified remote MSGQ.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_Put (IN  LdrvMsgqTransportHandle mqtHandle,
          IN  MsgQueueId              msgqId,
          IN  MsgqMsg                 msg) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	Handle to the transport object.	
IN	MsgQueueId	msgqId
	ID of the destination MSGQ.	
IN	MsgqMsg	msg
	Pointer to the message to be sent to the destination MSGQ.	

Return Values

DSP_SOK	The message has been successfully sent.
DSP_EINVALIDARG	Invalid argument.
DSP_ENOTFOUND	The message queue does not exist. This implies that the MSGQ has not been located before this call was made.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

msg must be valid.

Post Conditions

None

See Also

RMQT_PutCallback

21.8 RMQT_GetReplyId

This function extracts the MSGQ ID to be used for replying to a received message.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_GetReplyId (IN  LdrvMsgqTransportHandle mqtHandle,
                 IN  MsgqMsg                 msg,
                 OUT MsgQueueId *            msgqId) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	Handle to the transport object.	
IN	MsgqMsg	msg
	Message, whose reply MSGQ ID is to be extracted.	
OUT	MsgQueueId *	msgqId
	Location to receive the ID of the reply MSGQ.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msg must be valid.

msgqId must be valid.

Post Conditions

None

See Also

None

21.9 RMQT_PutCallback

This function implements the callback that runs when the message to be sent to a remote MSGQ has been transferred across the physical link.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_PutCallback (IN  ProcessorId   procId,
                  IN  DSP_STATUS    statusOfIo,
                  IN  Uint8 *       buffer,
                  IN  Uint32        size,
                  IN  Pvoid         mqtHandle) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	DSP_STATUS	statusOfIo
	Status of the IO requested.	
IN	Uint8 *	buffer
	Pointer to the message buffer.	
IN	Uint32	size
	Size of the message buffer.	
IN	Pvoid	mqtHandle
	Argument associated with the IO request.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

buffer must be valid.

mqtHandle must be valid.

Post Conditions

None

See Also

RMQT_Put

21.10 RMQT_GetCallback

This function implements the callback that runs when the message has been received from the DSP.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_GetCallback (IN  ProcessorId   procId,
                  IN  DSP_STATUS    statusOfIo,
                  IN  Uint8 *       buffer,
                  IN  Uint32        size,
                  IN  Pvoid         mqtHandle) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	DSP_STATUS	statusOfIo
	Status of the IO requested.	
IN	Uint8 *	buffer
	Pointer to the message buffer.	
IN	Uint32	size
	Size of the message buffer.	
IN	Pvoid	mqtHandle
	Argument associated with the IO request.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

procId must be valid.

buffer must be valid.

arg must be valid.

Post Conditions

None

See Also

RMQT_Open

21.11 RMQT_Instrument

This function gets the instrumentation information related to the specified message queue.

Syntax

```
NORMAL_API
DSP_STATUS
RMQT_Instrument (IN  LdrvMsgqTransportHandle mqtHandle,
                 IN  MsgQueueId              msgqId,
                 OUT MsgqInstrument *        retVal) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MQT transport object.	
IN	MsgQueueId	msgqId
	Message queue identifier.	
OUT	MsgqInstrument *	retVal
	Location to retrieve the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

retVal must be valid.

Post Conditions

None

See Also

None

21.12 RMQT_Debug

This function gets the instrumentation information related to the specified message queue.

Syntax

```
NORMAL_API
Void
RMQT_Debug (IN  LdrvMsgqTransportHandle mqtHandle, IN  MsgQueueId
msgqId) ;
```

Arguments

IN	LdrvMsgqTransportHandle	mqtHandle
	This is the handle to LDRV MQT transport object.	
IN	MsgQueueId	msgqId
	Message queue identifier.	

Return Values

None.

Pre Conditions

mqtHandle must be valid.

msgqId must be valid.

Post Conditions

None

See Also

None

DSP COMPONENT

22 dspdefs.h

Defines the DSP object structure and associated structures.

Some structures, already defined in CFG, have been redefined here to make the DSP subcomponent independent of CFG.

Path

$(DSPLINK)\gpp\inc

Revision

00.13

TYPE DEFINITIONS & STRUCTURES

22.1 DspMmuEntry

MMU Entry for the DSP.

Syntax

```
typedef struct DspMmuEntry_tag {
    Uint32  entry           ;
    Uint32  virtualAddress  ;
    Uint32  physicalAddress ;
    Uint32  size            ;
    Uint32  access          ;
    Uint32  preserve        ;
    Uint32  mapInGpp        ;
} DspMmuEntry ;
```

Fields

Uint32	entry Entry number for the MMU record.
Uint32	virtualAddress Virtual address.
Uint32	physicalAddress Physical address.
Uint32	size Indicates the size of MMU TLB entry.
Uint32	access Access permission.
Uint32	preserve Indicates whether entry is preserved or not.

Uint32	mapInGpp Flag indicating whether DSP address is mapped to GPP address space.

See Also

FUNCTIONS

22.2 FnDspSetup

Signature of function that sets up components to make DSP reachable from GPP.

Syntax

```
typedef DSP_STATUS (*FnDspSetup) (IN ProcessorId dspId,
                                  IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.

Pre Conditions

Post Conditions

See Also

22.3 FnDspInitialize

Signature of function that resets the DSP and initializes the components required by DSP.

Syntax

```
typedef DSP_STATUS (*FnDspInitialize) (IN ProcessorId dspId,
                                       IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.4 FnDspFinalize

Signature of function that resets the DSP.

Syntax

```
typedef DSP_STATUS (*FnDspFinalize) (IN ProcessorId dspId,
                                     IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.5 FnDspStart

Signature of function that causes DSP to start execution from the given DSP address.

Syntax

```
typedef DSP_STATUS (*FnDspStart) (IN ProcessorId dspId,
                                  IN DspObject * dspObj,
                                  IN Uint32      dspAddr) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Uint32	dspAddr
	Address to start execution from.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.6 FnDspStop

Signature of function that stops execution on DSP.

Syntax

```
typedef DSP_STATUS (*FnDspStop) (IN ProcessorId dspId,
                                 IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.7 FnDspIdle

Signature of function that idles the DSP.

Syntax

```
typedef DSP_STATUS (*FnDspIdle) (IN ProcessorId dspId,
                                 IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.8 FnDspEnableInterrupt

Signature of function that enables the specified interrupt for communication with DSP.

Syntax

```
typedef DSP_STATUS (*FnDspEnableInterrupt) (IN ProcessorId       dspId,
                                            IN DspObject *
dspObj,
                                            IN InterruptObject *
intInfo) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.9 FnDspDisableInterrupt

Signature of function that disables the specified interrupt for communication with DSP.

Syntax

```
typedef DSP_STATUS (*FnDspDisableInterrupt) (IN ProcessorId
dspId,
                                             IN DspObject *
dspObj,
                                             IN InterruptObject *
intInfo) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.10 FnDspInterrupt

Signature of function that sends the specified interrupt to DSP.

Syntax

```
typedef DSP_STATUS (*FnDspInterrupt) (IN ProcessorId       dspId,
                                      IN DspObject *       dspObj,
                                      IN InterruptObject * intInfo,
                                      IN Pvoid             arg) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	
IN	Pvoid	arg
	Value to send with the interrupt.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.11 FnDspClearInterrupt

Clear the DSP Interrupt.

Syntax

```
typedef DSP_STATUS (*FnDspClearInterrupt) (IN  ProcessorId       dspId,
                                           IN  DspObject *
dspObj,
                                           IN  InterruptObject *
intInfo,
                                           OUT Pvoid
retVal) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	
OUT	Pvoid	retVal
	Interrupt value present before clearing the interrupt.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.12 FnDspRead

Signature of function that reads data from DSP.

Syntax

```
typedef DSP_STATUS (*FnDspRead) (IN     ProcessorId  dspId,
                                 IN     DspObject *  dspObj,
                                 IN     Uint32       dspAddr,
                                 IN     Endianism    endianism,
                                 IN OUT Uint32 *     numBytes,
                                 OUT    Uint8 *      buffer) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Uint32	dspAddr
	DSP address to read from.	
IN	Endianism	endianism
	endianness of data - indicates whether swap is required or not.	
IN OUT	Uint32 *	numBytes
	Number of bytes to read.	
OUT	Uint8 *	buffer
	Buffer to hold the read data.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.13 FnDspWrite

Signature of function that writes data to DSP.

Syntax

```
typedef DSP_STATUS (*FnDspWrite) (IN ProcessorId dspId,
                                  IN DspObject * dspObj,
                                  IN Uint32      dspAddr,
                                  IN Endianism   endianism,
                                  IN Uint32      numBytes,
                                  IN Uint8 *     buffer) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Uint32	dspAddr
	DSP address to write to.	
IN	Endianism	endianism
	endianness of data - indicates whether swap is required or not.	
IN	Uint32	numBytes
	Number of bytes to write.	
IN	Uint8 *	buffer
	Buffer containing data to be written.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	dspId is invalid.
DSP_EPOINTER	DSP_Setup function wasn't called before calling this function.

Pre Conditions

Post Conditions

See Also

22.14 FnDspControl

Hook for performing device dependent control operation.

Syntax

```
typedef DSP_STATUS (*FnDspControl) (IN  ProcessorId dspId,
                                    IN  DspObject * dspObj,
                                    IN  Int32       cmd,
                                    OPT Pvoid       arg) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid arguments specified.

Pre Conditions

Post Conditions

See Also

22.15 FnDspInstrument

Gets the instrumentation information related to the specified DSP object.

Syntax

```
typedef DSP_STATUS (*FnDspInstrument) (IN DspObject * dspObj,
                                       OUT DspStats * retVal) ;
```

Arguments

IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
OUT	DspStats *	retVal
	Placeholder to return the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument(s).

Pre Conditions

Post Conditions

See Also

22.16 FnDspDebug

Prints debug information of the specified DSP object.

Syntax

```
typedef Void (*FnDspDebug) (IN DspObject * dspObj) ;
```

Arguments

IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

Post Conditions

See Also

22.17 DspInterface

Interface functions exported by the DSP subcomponent.

Syntax

```
typedef struct DspInterface_tag {
    FnDspSetup              setup            ;
    FnDspInitialize         initialize       ;
    FnDspFinalize           finalize         ;
    FnDspStart              start            ;
    FnDspStop               stop             ;
    FnDspIdle               idle             ;
    FnDspEnableInterrupt    enableInterrupt  ;
    FnDspDisableInterrupt   disableInterrupt ;
    FnDspInterrupt          interrupt        ;
    FnDspClearInterrupt     clearInterrupt   ;
    FnDspRead               read             ;
    FnDspWrite              write            ;
    FnDspControl            control          ;
#if defined (DDSP_PROFILE)
    FnDspInstrument         instrument       ;
#endif
#if defined (DDSP_DEBUG)
    FnDspDebug              debug            ;
#endif
} DspInterface ;
```

Fields

FnDspSetup	setup Function pointer to setup function for the DSP.
FnDspInitialize	initialize Function pointer to initialize function for the DSP.
FnDspFinalize	finalize Function pointer to finalize function for the DSP.
FnDspStart	start Function pointer to start function for the DSP.
FnDspStop	stop Function pointer to stop function for the DSP.
FnDspIdle	idle Function pointer to idle function for the DSP.

FnDspEnableInterrupt	enableInterrupt Function pointer to enableInterrupt function for the DSP.
FnDspDisableInterrupt	disableInterrupt Function pointer to disableInterrupt function for the DSP.
FnDspInterrupt	interrupt Function pointer to interrupt function for the DSP.
FnDspClearInterrupt	clearInterrupt Function pointer to clearInterrupt function for the DSP.
FnDspRead	read Function pointer to read function for the DSP.
FnDspWrite	write Function pointer to write function for the DSP.
FnDspControl	control Function pointer to perform device dependent control operation.
FnDspInstrument	instrument Function pointer to instrument function for the DSP.
FnDspDebug	debug Function pointer to debug function for the DSP.

See Also

22.18 LoaderInterface

Structure containing interface functions exported by the loader subcomponent.

Syntax

```
typedef DSP_STATUS (*FnLoad) (IN  ProcessorId     dspId,
                              IN  LoaderObject *  loaderObj,
                              IN  Uint32          argc,
                              IN  Char8 **        argv,
                              OUT Uint32 *        entryPt) ;

typedef DSP_STATUS (*FnLoadSection) (IN  ProcessorId     dspId,
                                     IN  LoaderObject *  loaderObj,
                                     IN  Uint32          sectId) ;


typedef struct LoaderInterface_tag {
    FnLoad         load         ;
    FnLoadSection  loadSection  ;
} LoaderInterface ;
```

Fields

FnLoad	load Function pointer providing the abstraction to the loader's load module.
FnLoadSection	loadSection Function pointer providing the abstraction to the loader's loadSection module.

See Also

22.19 DspObject

DSP object.

Syntax

```
struct DspObject_tag {
    Char8              dspName    [DSP_MAX_STRLEN] ;
    DspArch            dspArch                     ;
    Char8              execName   [DSP_MAX_STRLEN] ;
    LoaderInterface  * loaderInterface             ;
#if defined (CHNL_COMPONENT)
    LinkAttrs *        linkTable                   ;
    Uint32             numLinks                    ;
#endif
    Uint32             autoStart                   ;
    Uint32             resetVector                 ;
    Uint32             wordSize                    ;
    Uint32             endian                      ;
    Bool               mmuFlag                     ;
    DspMmuEntry *      mmuTable                    ;
    Uint32             numMmuEntries               ;
    DspInterface *     interface                   ;
    Uint32 *           addrMapInGpp                ;
#if defined (MSGQ_COMPONENT)
    Uint32             mqtId                       ;
#endif
#if defined (DDSP_PROFILE)
    DspStats *         dspStats                    ;
#endif
} ;
```

Fields

Char8	dspName Name of the DSP.
DspArch	dspArch Architecture of the Dsp.
Char8	execName Name of default DSP executable.
LoaderInterface *	loaderInterface The function pointer interface to access the services of the loader subcomponent for this DSP.
LinkAttrs *	linkTable Array of link attributes.

Uint32	numLinks Number of links towards the DSP.
Uint32	autoStart Auto start flag for the DSP.
Uint32	resetVector Reset vector address for the dsp.
Uint32	wordSize Word size of the DSP.
Uint32	endian Endianism of the DSP.
Bool	mmuFlag Indicates if the MMU is enabled on the DSP.
DspMmuEntry *	mmuTable Table of MMU entries.
Uint32	numMmuEntries Number of MMU entries.
DspInterface *	interface The function pointer interface to access the services of the DSP subcomponent for this DSP.
Uint32 *	addrMapInGpp Array holding GPP address corresponding to DSP address. Size of the array is 'numMmuEntries' to hold entries corresponding to the number of entries in DSP MMU table. The value at an index will be ADDRMAP_INVALID in case no mapping is required corresponding to a DSP MMU entry.
Uint32	mqtId The id of the MQT which is to be used for this DSP.
DspStats *	dspStats Profiling information related to the target DSP.

See Also

22.20 LoaderInterface

Structure containing interface functions exported by the loader subcomponent.

Syntax

```
typedef DSP_STATUS (*FnLoad) (IN  ProcessorId     dspId,
                              IN  LoaderObject *  loaderObj,
                              IN  Uint32          argc,
                              IN  Char8 **        argv,
                              OUT Uint32 *        entryPt) ;

typedef DSP_STATUS (*FnLoadSection) (IN  ProcessorId     dspId,
                                     IN  LoaderObject *  loaderObj,
                                     IN  Uint32          sectId) ;


typedef struct LoaderInterface_tag {
    FnLoad         load         ;
    FnLoadSection  loadSection  ;
} LoaderInterface ;
```

Arguments

IN	ProcessorId	dspId
	Target DSP identifier where the section is to be loaded.	
IN	LoaderObject *	loaderObj
	This object is used to receive arguments from PMGR.	
IN	Uint32	sectId
	Identifier for section to load.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFILE	Invalid base image
DSP_EACCESSDENIED	Not allowed to access the DSP
DSP_EFAIL	General failure, unable to load image onto DSP
DSP_EINVALIDARG	Invalid dspId argument.
DSP_SOK	Operation successfully completed.
DSP_EFILE	Invalid base image
DSP_EACCESSDENIED	Not allowed to access the DSP

DSP_EFAIL	General failure, unable to load image onto DSP
DSP_EINVALIDARG	Invalid dspId argument.
DSP_EINVALIDSECT	Invalid section name.

See Also

23 dsp.h

Defines interface exposed by DSP sub-component.

Path

`$(DSPLINK)\gpp\src\ldrv`

Revision

`00.10`

FUNCTIONS

23.1 DSP_Setup

Sets up components to make DSP reachable from GPP.

Causes no state transition on DSP.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Setup (IN ProcessorId dspId, IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid.

Pre Conditions

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

`DSP_Initialize`

23.2 DSP_Initialize

Resets the DSP and initializes the components required by DSP.

Puts the DSP in RESET state.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Initialize (IN ProcessorId  dspId, IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

DSP_Setup
DSP_Finalize

23.3 DSP_Finalize

Resets the DSP and puts it into IDLE Mode.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Finalize (IN ProcessorId dspId, IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

DSP_Initialize

23.4 DSP_Start

Causes DSP to start execution from the given DSP address.

DSP is put to STARTED state after successful completion.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Start (IN ProcessorId dspId, IN DspObject * dspObj, IN Uint32
dspAddr) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Uint32	dspAddr
	Address to start execution from.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

DSP_Stop

23.5 DSP_Stop

Stops execution on DSP.

DSP is put to STOPPED state after successful completion.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Stop (IN ProcessorId dspId, IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

DSP_Start

23.6 DSP_Idle

Idles the DSP.

DSP is put to IDLE state after successful completion.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Idle (IN ProcessorId dspId, IN DspObject * dspObj) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

DSP_Start

23.7 DSP_EnableInterrupt

Enables the specified interrupt for communication with DSP.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_EnableInterrupt (IN ProcessorId         dspId,
                     IN DspObject *         dspObj,
                     IN InterruptObject *   intInfo) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

```
DSP_Setup
DSP_Interrupt
DSP_DisableInterrupt
```

23.8 DSP_DisableInterrupt

Disables the specified interrupt for communication with DSP.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_DisableInterrupt (IN ProcessorId        dspId,
                      IN DspObject *        dspObj,
                      IN InterruptObject *  intInfo) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Id.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

```
DSP_Setup
DSP_Interrupt
DSP_EnableInterrupt
```

23.9 DSP_Interrupt

Sends the specified interrupt to DSP.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Interrupt (IN     ProcessorId       dspId,
               IN     DspObject *       dspObj,
               IN     InterruptObject * intInfo,
               IN OPT Pvoid             arg) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	
IN OPT	Pvoid	arg
	Pointer to a value to send with the interrupt.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid.
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

DSP_Start

23.10 DSP_ClearInterrupt

Clear the DSP Interrupt.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_ClearInterrupt (IN  ProcessorId         dspId,
                    IN  DspObject *         dspObj,
                    IN  InterruptObject *   intInfo,
                    OUT Pvoid               retVal) ;
```

Arguments

IN	ProcessorId	dspId
	Processor Identifier.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	InterruptObject *	intInfo
	Pointer to object containing interrupt information.	
OUT	Pvoid	retVal
	Interrupt value present before clearing the interrupt.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

retval must be a valid pointer.

Post Conditions

None.

See Also

DSP_Start

23.11 DSP_Read

Reads data from DSP.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Read (IN     ProcessorId  dspId,
          IN     DspObject *  dspObj,
          IN     Uint32       dspAddr,
          IN     Endianism    endianInfo,
          IN OUT Uint32 *     numBytes,
          OUT    Uint8 *      buffer) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Uint32	dspAddr
	DSP address to read from.	
IN	Endianism	endianInfo
	endianness of data - indicates whether swap is required or not.	
IN OUT	Uint32 *	numBytes
	Number of bytes to read.	
OUT	Uint8 *	buffer
	Buffer to hold the read data.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

numBytes must be a valid pointer.

buffer must be a valid pointer.

Post Conditions

On successful completion, holds the number of bytes read.

See Also

None.

23.12 DSP_Write

Writes data to DSP.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Write (IN ProcessorId dspId,
           IN DspObject * dspObj,
           IN Uint32      dspAddr,
           IN Endianism   endianInfo,
           IN Uint32      numBytes,
           IN Uint8 *     buffer) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Uint32	dspAddr
	DSP address to write to.	
IN	Endianism	endianInfo
	endianness of data - indicates whether swap is required or not.	
IN	Uint32	numBytes
	Number of bytes to write.	
IN	Uint8 *	buffer
	Buffer containing data to be written.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Arguments dspId and/or dspObj are invalid..
DSP_EFAIL	DSP_Setup function wasn't called before calling this function.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

buffer must be a valid pointer.

Post Conditions

None.

See Also

None.

23.13 DSP_Control

Hook for performing device dependent control operation.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Control (IN  ProcessorId dspId,
             IN  DspObject * dspObj,
             IN  Int32       cmd,
             OPT Pvoid       arg) ;
```

Arguments

IN	ProcessorId	dspId
	Processor ID.	
IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
IN	Int32	cmd
	Command id.	
OPT	Pvoid	arg
	Optional argument for the specified command.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid arguments specified.

Pre Conditions

DSP_Setup must be called before calling this function.

dspId must be a valid DSP identifier.

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

None.

23.14 DSP_Instrument

Gets the instrumentation information related to the specified DSP object.

Syntax

```
NORMAL_API
DSP_STATUS
DSP_Instrument (IN DspObject * dspObj, OUT DspStats * retVal) ;
```

Arguments

IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	
OUT	DspStats *	retVal
	Placeholder to return the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument(s).

Pre Conditions

dspObj must be valid.

retVal must be valid.

Post Conditions

None.

See Also

DspObject

23.15 DSP_Debug

Prints debug information of the specified DSP object.

Syntax

```
NORMAL_API
Void
DSP_Debug (IN DspObject * dspObj) ;
```

Arguments

IN	DspObject *	dspObj
	Pointer to object containing context information for DSP.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

dspObj must be pointing to a valid DSP object.

Post Conditions

None.

See Also

None.

OSAL COMPONENT

24 cfgdefs.h

Defines the global data structure for CFG.

Path

`$(DSPLINK)\gpp\inc`

Revision

`00.10`

CONSTANTS

24.1 CFG_MAX_STRLEN

Maximum length of the strings used in CFG.

Syntax

```
#define CFG_MAX_STRLEN  32
```

See Also

TYPE DEFINITIONS & STRUCTURES

24.2 CFG_Driver

Driver configuration structure.

Syntax

```
typedef struct CFG_Driver_tag {
    Char8  driverName [CFG_MAX_STRLEN]    ;
    Uint32 components                     ;
    Uint32 queueLength                    ;
    Uint32 linkTables                     ;
    Uint32 mmuTables                      ;
#if defined (MSGQ_COMPONENT)
    Uint32 numMqas                        ;
    Uint32 numMqts                        ;
    Uint32 localMqt                       ;
#endif
} CFG_Driver ;
```

Fields

Char8	driverName Name of driver.
Uint32	components Number of components of driver.
Uint32	queueLength Length of queue supported by driver.
Uint32	linkTables Number of Link tables in "this" configuration.
Uint32	mmuTables Number of MMU tables in "this" configuration.
Uint32	numMqas Number of MQA's for messaging.
Uint32	numMqts Number of MQA's for messaging.
Uint32	localMqt The id of the MQT which is to be used as Local MQT.

See Also

24.3 CFG_Gpp

Driver configuration structure.

Syntax

```
typedef struct CFG_Gpp_tag {
    Char8   gppName [CFG_MAX_STRLEN] ;
    Uint32  numDsps ;
} CFG_Gpp ;
```

Fields

Char8	gppName Name of GPP Processor.
Uint32	numDsps Number of DSPs.

See Also

24.4 CFG_Dsp

Processor configuration structure.

Syntax

```
typedef struct CFG_Dsp_tag {
    Char8   dspName    [CFG_MAX_STRLEN] ;
    Uint32  dspArch          ;
    Char8   execName   [CFG_MAX_STRLEN] ;
    Pvoid   loaderInterface ;
    Uint32  linkTable        ;
    Uint32  linkTableSize    ;
    Uint32  autoStart        ;
    Uint32  resetVector      ;
    Uint32  wordSize         ;
    Uint32  endian           ;
    Uint32  mmuFlag          ;
    Uint32  mmuTable         ;
    Uint32  mmuTableSize     ;
    Pvoid   interface        ;
#if defined (MSGQ_COMPONENT)
    Uint32  mqtId            ;
#endif
} CFG_Dsp ;
```

Fields

Char8	dspName Name of DSP processor.
Uint32	dspArch Architecture of the DSP.
Char8	execName Name of executable to load.
Pvoid	loaderInterface Function pointer interface for accessing the loader.
Uint32	linkTable Table number of the link(s) toward this DSP
Uint32	linkTableSize Size of the link table.
Uint32	autoStart

	AutoStart flag.
Uint32	resetVector Address of reset vector of DSP.
Uint32	wordSize Word size of DSP in bytes.
Uint32	endian Endian info of DSP.
Uint32	mmuFlag Is MMU used?
Uint32	mmuTable Table number of the MMU entries for this DSP.
Uint32	mmuTableSize Number of entries in MMU table.
Pvoid	interface Function pointer interface for accessing the DSP.
Uint32	mqtId The id of the MQT which is to be used for this DSP.

See Also

24.5 CFG_Link

Link configuration structure.

Syntax

```
typedef struct CFG_Link_tag {
    Char8   linkName [CFG_MAX_STRLEN] ;
    Char8   abbr     [CFG_MAX_STRLEN] ;
    Uint32  baseChnlId ;
    Uint32  numChannels ;
    Uint32  maxBufSize ;
    Pvoid   interfaceTable ;
    Uint32  argument1 ;
    Uint32  argument2 ;
} CFG_Link ;
```

Fields

Char8	linkName Name of Link.
Char8	abbr Abbreviation of the link name.
Uint32	baseChnlId Base channel ID for this link.
Uint32	numChannels Number of channels for this link.
Uint32	maxBufSize Maximum size of data buffer on this link.
Pvoid	interfaceTable Interface function table address.
Uint32	argument1 Link specific argument 1. The significance of this argument is specific to a link driver.
Uint32	argument2 Link specific argument 2. The significance of this argument is specific to a link driver.

See Also

24.6 CFG_MmuEntry

Defines an entry in the MMU table.

Syntax

```
typedef struct CFG_MmuEntry_tag {
    Uint32  entry           ;
    Uint32  virtualAddress  ;
    Uint32  physicalAddress ;
    Uint32  size            ;
    Uint32  access          ;
    Uint32  preserve        ;
    Uint32  mapInGpp        ;
} CFG_MmuEntry ;
```

Fields

Uint32	entry Entry number.
Uint32	virtualAddress virtual address field of entry.
Uint32	physicalAddress physical address field of entry.
Uint32	size Size field of entry.
Uint32	access Access Permission.
Uint32	preserve Preserve field of entry.
Uint32	mapInGpp Flag indicating whether DSP address is mapped to GPP address space.

See Also

24.7 CFG_Mqa

This structure defines the MQA configuration structure.

Syntax

```
typedef struct CFG_Mqa_tag {
    Char8  mqaName    [CFG_MAX_STRLEN] ;
    Pvoid  interface                   ;
} CFG_Mqa ;
```

Fields

Char8	mqaName Name of the MQA. For debugging purposes only.
Pvoid	interface Function pointer interface to access the functions for this MQA.

See Also

24.8 CFG_Mqt

This structure defines the MQT configuration structure.

Syntax

```
typedef struct CFG_Mqt_tag {
    Char8  mqtName    [CFG_MAX_STRLEN] ;
    Pvoid  interface                   ;
    Uint32 linkId                      ;
} CFG_Mqt ;
```

Fields

Char8	mqtName Name of the MQT. For debugging purposes only.
Pvoid	interface Function pointer interface to access the functions for this MQT.
Uint32	linkId ID of the link used by this MQT.

See Also

25 osal.h

Defines the interfaces for initializing and finalizing OSAL.

Path

`$(DSPLINK)\gpp\src\osal`

Revision

`00.03`

FUNCTIONS

25.1 OSAL_Initialize

Initializes the OS Adaptation layer.

Syntax

```
EXPORT_API
DSP_STATUS
OSAL_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

None

Post Conditions

None

See Also

OSAL_Finalize

25.2 OSAL_Finalize

Releases OS adaptation layer resources indicating that they would no longer be used.

Syntax

```
EXPORT_API
DSP_STATUS
OSAL_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

Subcomponent must be initialized.

Post Conditions

None

See Also

OSAL_Initialize

26 cfg.h

Defines the interface and data structures for the CFG subcomponent.

Path

$(DSPLINK)\gpp\src\osal

Revision

00.07

CONSTANTS

26.1 CFG_DRIVER_BASE

Base of the keys to fetch driver related information from the configuration database.

Syntax

```
#define CFG_DRIVER_BASE         (Uint32) 0x0000
```

See Also

26.2 CFG_GPP_BASE

Base of the keys to fetch GPP related information from the configuration database.

Syntax

```
#define CFG_GPP_BASE            (Uint32) 0x1000
```

See Also

26.3 CFG_DSP_BASE

Base of the keys to fetch DSP related information from the configuration database.

Syntax

```
#define CFG_DSP_BASE            (Uint32) 0x2000
```

See Also

26.4 CFG_LINK_BASE

Base of the keys to fetch link related information from the configuration database.

Syntax

```
#define CFG_LINK_BASE           (Uint32) 0x3000
```

See Also

26.5 CFG_MMU_BASE

Base of the keys to fetch MMU related information from the configuration database.

Syntax

```
#define CFG_MMU_BASE            (Uint32) 0x4000
```

See Also

26.6 CFG_MQA_BASE

Base of the keys to fetch MQA related information from the configuration database.

Syntax

```
#define CFG_MQA_BASE            (Uint32) 0x5000
```

See Also

26.7 CFG_MQT_BASE

Base of the keys to fetch MQT related information from the configuration database.

Syntax

```
#define CFG_MQT_BASE            (Uint32) 0x6000
```

See Also

26.8 CFG_ID_LAST

Last ID used.

Syntax

```
#define CFG_ID_LAST             (Uint32) 0x4FFF
```

See Also

26.9 CFG_ID_NONE

Identifier value used when no ID is associated to the key whose value is being requested.

Syntax

```
#define CFG_ID_NONE             (Uint32) 0xFFFFFFFF
```

See Also

FUNCTIONS

26.10 CFG_Initialize

This function initializes this sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
CFG_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation Successful.
DSP_EMEMORY	Out of memory error.
DSP_EFAIL	Operation Failed.

Pre Conditions

None

Post Conditions

None

See Also

CFG_Finalize

26.11 CFG_Finalize

This function provides an interface to exit from this sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
CFG_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation Successful.
DSP_EMEMORY	Out of memory error.
DSP_EFAIL	Operation Failed.

Pre Conditions

None

Post Conditions

None

See Also

CFG_Initialize

26.12 CFG_GetRecord

Gets the record from the configuration.

Syntax

```
EXPORT_API
DSP_STATUS
CFG_GetRecord (IN Uint32 key, IN Uint32 id, OUT Void * record) ;
```

Arguments

IN	Uint32	key
	Key for configuration item.	
IN	Uint32	id
	Context dependent identifier. The meaning of this argument depends upon the given key.	
OUT	Void *	record
	Location where record is to be stored.	

Return Values

DSP_SOK	Operation Successful.
DSP_EINVALIDARG	Invalid arguments.
DSP_EFAIL	Operation Failed.

Pre Conditions

record must be a valid pointer.

Post Conditions

None

See Also

None

26.13 CFG_GetNumValue

Gets the numeric value type of configuration.

Syntax

```
EXPORT_API
DSP_STATUS
CFG_GetNumValue (IN Uint32 key, IN Uint32 id, OUT Uint32 * value) ;
```

Arguments

IN	Uint32	key
	Key for configuration item.	
IN	Uint32	id
	Context dependent identifier. The meaning of this argument depends upon the given key.	
OUT	Uint32 *	value
	Location where value is to be stored.	

Return Values

DSP_SOK	Operation Successful.
DSP_EINVALIDARG	Invalid arguments.
DSP_EFAIL	Operation Failed.

Pre Conditions

value must be a valid pointer.

Post Conditions

None

See Also

None

26.14 CFG_GetStrValue

Gets the string value type of configuration.

Syntax

```
EXPORT_API
DSP_STATUS
CFG_GetStrValue (IN Uint32 key, IN Uint32 id, OUT Pstr  string) ;
```

Arguments

IN	Uint32	key
	Key for configuration item.	
IN	Uint32	id
	Context dependent identifier. The meaning of this argument depends upon the given key.	
OUT	Pstr	string
	Location where string is to be stored.	

Return Values

DSP_SOK	Operation Successful.
DSP_EINVALIDARG	Invalid arguments.
DSP_EFAIL	Operation Failed.

Pre Conditions

string must be a valid pointer.

Post Conditions

None

See Also

None

27 dpc.h

Defines the interfaces and data structures for the sub-component DPC.

Path

$(DSPLINK)\gpp\src\osal

Revision

00.07

FUNCTIONS

27.1 FnDpcProc

Function prototype for DPC function. The user defined functions that is to be invoked as a DPC should conform to this signature.

Syntax

```
typedef Void (*FnDpcProc) (Pvoid refData) ;
```

Arguments

Pvoid	refData
Argument to be passed to DPC call.	

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

DPC_Callback
DPC_Create

27.2 DPC_Initialize

Initializes the DPC module. It initializes the global area for holding all the DPC objects.

Syntax

```
EXPORT_API
DSP_STATUS
DPC_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.

Pre Conditions

None

Post Conditions

DPC must be initialized.

See Also

DPC_Finalize

27.3 DPC_Finalize

Releases all resources used by this sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
DPC_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.

Pre Conditions

DPC must be initialized.

Post Conditions

All in-use DPC objects are released.

See Also

DPC_Initialize

27.4 DPC_Create

Creates a DPC object and returns it after populating relevant fields.

Syntax

```
EXPORT_API
DSP_STATUS
DPC_Create (IN  FnDpcProc       userDPCFn,
            IN  Pvoid           dpcArgs,
            OUT DpcObject **    dpcObj) ;
```

Arguments

IN	FnDpcProc	userDPCFn
	User specified function to be invoked as DPC.	
IN	Pvoid	dpcArgs
	Arguments to be passed to the DPC.	
OUT	DpcObject **	dpcObj
	OUT argument to store the DPC object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_ERESOURCE	No more DPC objects are available for creation.
DSP_EINVALIDARG	Invalid arguments.

Pre Conditions

DPC must be initialized.

userDPCFn must be a valid function.

dpcObj must be a valid pointer.

Post Conditions

*dpcObj points to an initialized DPC Object on successful

completion or *dpcObj is NULL on failure.

See Also

DPC_Delete
DPC_Schedule

27.5 DPC_Delete

Deletes the DPC object.

Syntax

```
EXPORT_API
DSP_STATUS
DPC_Delete (IN DpcObject * dpcObj) ;
```

Arguments

IN	DpcObject *	dpcObj
	The DPC object to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid dpcObj object.
DSP_EINVALIDARG	Incorrect dpcObj specified.

Pre Conditions

DPC must be initialized.

dpcObj must be a valid DPC object.

Post Conditions

Upon successful completion the dpcObj is reset.

See Also

DPC_Create

27.6 DPC_Cancel

Cancels any pending DPCs associated to dpcObj.

Syntax

```
EXPORT_API
DSP_STATUS
DPC_Cancel (IN DpcObject * dpcObj) ;
```

Arguments

IN	DpcObject *	dpcObj
	The DPC object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid dpcObj object.
DSP_EINVALIDARG	Incorrect dpcObj specified.

Pre Conditions

DPC must be initialized.

dpcObj must be a valid DPC object.

Post Conditions

All pending calls to the DPC are cancelled.

See Also

DPC_Schedule
DPC_Create

27.7 DPC_Schedule

Schedules the user defined function associated with dpcObj to be invoked at a later point of time.

Syntax

```
EXPORT_API
DSP_STATUS
DPC_Schedule (IN DpcObject * dpcObj) ;
```

Arguments

IN	DpcObject *	dpcObj
	The DPC object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid dpcObj object.
DSP_EINVALIDARG	Incorrect dpcObj specified.

Pre Conditions

DPC must be initialized.

dpcObj must be a valid DPC object.

Post Conditions

None

See Also

```
DPC_Create
DPC_Cancel
DPC_Callback
```

27.8 DPC_Disable

Disables execution of DPCs.

Syntax

```
EXPORT_API
Void
DPC_Disable () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

27.9 DPC_Enable

Enables execution of DPCs.

Syntax

```
EXPORT_API
Void
DPC_Enable () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

27.10 DPC_Debug

Prints the current status of DPC objects.

Syntax

```
EXPORT_API
Void
DPC_Debug () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

28 isr.h

Defines the interfaces and data structures for the sub-component ISR.

Path

$(DSPLINK)\gpp\src\osal

Revision

00.07

ENUMERATIONS

28.1 ISR_State

Enumerates the various states of ISR.

Syntax

```
typedef enum {
    ISR_Installed   = 0,
    ISR_Uninstalled = 1,
    ISR_Disabled    = 2,
    ISR_Enabled     = 3
} ISR_State ;
```

Enum Values

ISR_Installed	Indicates that the ISR is installed.
ISR_Uninstalled	Indicates that the ISR is uninstalled.
ISR_Disabled	Indicates that the ISR is disabled.
ISR_Enabled	Indicates that the ISR is enabled.

See Also

FUNCTIONS

28.2 ISR_Initialize

Forward declaration for IsrObject, actual definition is OS dependent.

Function prototype for an ISR. The user defined function to be invoked as an ISR should conform to this signature.

Initializes and allocates resources used by ISR subcomponent.

Syntax

```
typedef struct IsrObject_tag IsrObject ;

typedef Void (*IsrProc) (Pvoid refData) ;


EXPORT_API
DSP_STATUS
ISR_Initialize () ;
```

Arguments

Pvoid	refData
Data to be passed to ISR when invoked.	

Return Values

None	
DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory.

Pre Conditions

None

Post Conditions

ISR must be initialized.

See Also

ISR_Install
ISR_Finalize

28.3 ISR_Finalize

Releases resources reserved for ISR subcomponent.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory.

Pre Conditions

ISR must be initialized.

Post Conditions

None

See Also

ISR_Initialize

28.4 ISR_Create

Creates an ISR object.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_Create (IN  IsrProc           fnISR,
            IN  Pvoid             refData,
            IN  InterruptObject * intObj,
            OUT IsrObject **      isrObj) ;
```

Arguments

IN	IsrProc	fnISR
	User defined interrupt service routine.	
IN	Pvoid	refData
	Argument to be passed to ISR when it is invoked.	
IN	InterruptObject *	intObj
	Interrupt information (OS and hardware dependent).	
OUT	IsrObject **	isrObj
	Out argument for IsrObject.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid arguments were passed to function.
DSP_EMEMORY	Out of memory error.

Pre Conditions

ISR must be initialized.

isrObj must be valid pointer.

intObj must be a valid pointer.

fnISR must be a valid function pointer.

Post Conditions

A valid IsrObject is returned on success.

See Also

ISR_Delete

28.5 ISR_Delete

Deletes the isrObject.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_Delete (IN IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	isrObj
	Object to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid isrObj object.
DSP_EMEMORY	Free memory error.
DSP_EACCESSDENIED	isrObj not uninstalled.

Pre Conditions

ISR must be initialized.

isrObj must be a valid object.

Post Conditions

None

See Also

ISR_Create

28.6 ISR_Install

Installs an interrupt service routine defined by isrObj.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_Install (IN  Void *       hostConfig,
             IN  IsrObject *  isrObj) ;
```

Arguments

IN	Void *	hostConfig
	Void pointer containing installation information related to installation of an ISR.	
IN	IsrObject *	isrObj
	The isrObj object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid isrObj object.
DSP_EACCESSDENIED	An ISR is already installed.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

ISR must be initialized.

isrObj must be valid.

Post Conditions

The isr is installed on success.

isrObj contains a valid IsrObject.

See Also

ISR_Func
ISR_Uninstall

28.7 ISR_Uninstall

Uninstalls the interrupt service routine defined by isrObj.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_Uninstall (IN IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	isrObj
	ISR object to uninstall.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid isrObj object.
DSP_EACCESSDENIED	ISR is already uninstalled.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

ISR must be initialized.

isrObj must be a valid IsrObject.

Post Conditions

None

See Also

ISR_Install

28.8 ISR_Disable

Disables an ISR associated with interrupt Id of isrObject.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_Disable (IN  IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	isrObj
	ISR object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EACCESSDENIED	ISR is not installed.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

ISR must be initialized.

isrObj must be a valid object.

Post Conditions

None

See Also

ISR_Enable
ISR_Install

28.9 ISR_Enable

Reactivates ISRs based on the specified flags argument. The flags argument must be obtained with an earlier call to ISR_Disable.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_Enable (IN  IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	isrObj
	ISR object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EACCESSDENIED	ISR is not installed.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

ISR must be initialized.

isrObj must be a valid object.

Post Conditions

None

See Also

ISR_Disable

28.10 ISR_GetState

Gets the state of an ISR.

Syntax

```
EXPORT_API
DSP_STATUS
ISR_GetState (IN  IsrObject *  isrObj,
              OUT ISR_State *  isrState) ;
```

Arguments

IN	IsrObject *	isrObj
	The ISR object.	
OUT	ISR_State *	isrState
	Current status of the ISR.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid isrObj object.
DSP_EINVALIDARG	Invalid isrStatus pointer.

Pre Conditions

isrObj must be a valid IsrObject.

isrStatus must be a valid pointer.

Post Conditions

None

See Also

```
ISR_Install
ISR_Uninstall
ISR_Enable
ISR_Disable
```

28.11 ISR_Debug

Prints the current status of ISR objects.

Syntax

```
EXPORT_API
Void
ISR_Debug () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

29 kfile.h

Defines interfaces and data structures for the sub-component KFILE.

This subcomponent assumes that a file system is available on the target platform.

Path

`$(DSPLINK)\gpp\src\osal`

Revision

`00.06`

ENUMERATIONS

29.1 KFILE_Seek

Enumerates the values used for repositioning the file position indicator.

Syntax

```
typedef enum {
    KFILE_SeekSet = 0x00,
    KFILE_SeekCur = 0x01,
    KFILE_SeekEnd = 0x02
} KFILE_FileSeek ;
```

Enum Values

KFILE_SeekSet	Seek from beginning of file.
KFILE_SeekCur	Seek from current position.
KFILE_SeekEnd	Seek from end of file.

See Also

FUNCTIONS

29.2 KFILE_Initialize

This function allocates and initializes all resources used by this subcomponent.

Syntax

```
EXPORT_API
DSP_STATUS
KFILE_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory.

Pre Conditions

None

Post Conditions

None

See Also

KFILE_Finalize

29.3 KFILE_Finalize

Releases resources used by this sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
KFILE_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory.

Pre Conditions

Subcomponent must be initialized.

Post Conditions

None

See Also

KFILE_Initialize

29.4 KFILE_Open

Opens a file.

Syntax

```
EXPORT_API
DSP_STATUS
KFILE_Open (IN CONST FileName       fileName,
            IN CONST Char8 *        mode,
            OUT      KFileObject ** fileHandle) ;
```

Arguments

IN CONST	FileName	fileName
	Name of the file to be opened.	
IN CONST	Char8 *	mode
	Mode for opening the file. This argument is case-sensitive. Expected modes are: "r" for read, "w" for write and "a" for append.	
OUT	KFileObject **	fileHandle
	Placeholder to return the file handle.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid arguments.
DSP_EFILE	File not found.
DSP_EMEMORY	Out of memory error.

Pre Conditions

Subcomponent must be initialized.

fileName must be valid.

mode must be valid.

fileHandle must be valid.

Post Conditions

fileHandle contains the fileObject on success.

See Also

KFILE_Close

29.5 KFILE_Close

Closes a file handle.

Syntax

```
EXPORT_API
DSP_STATUS
KFILE_Close (IN  KFileObject * fileObj) ;
```

Arguments

IN	KFileObject *	fileObj
	Handle of file to be closed, returned from KFILE_Open.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFILE	File is not open.
DSP_EPOINTER	Invalid file object.

Pre Conditions

Subcomponent must be initialized.

fileObj must be a valid handle to a file opened earlier.

Post Conditions

Memory allocated for fileObj is freed.

See Also

KFILE_Open

29.6 KFILE_Read

Reads a specified number of items of specified size (in bytes) to a buffer.

Syntax

```
EXPORT_API
DSP_STATUS
KFILE_Read (OUT Char8 *       buffer,
            IN  Uint32        size,
            IN  Uint32        count,
            IN  KFileObject * fileObj) ;
```

Arguments

OUT	Char8 *	buffer
	Buffer in which the contents of file are read.	
IN	Uint32	size
	Size of each object to read from file.	
IN	Uint32	count
	Number of objects to read.	
IN	KFileObject *	fileObj
	KFileObject to read from.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid arguments.
DSP_EPOINTER	Invalid file object.
DSP_EFILE	File is not open or error reading file.
DSP_ERANGE	The requested number of bytes is beyond EOF.

Pre Conditions

Subcomponent must be initialized.

fileObj must be a valid KFileObject pointer opened earlier.

Post Conditions

None

See Also

KFILE_Open

29.7 KFILE_Seek

Repositions the file pointer according to specified arguments.

Syntax

```
EXPORT_API
DSP_STATUS
KFILE_Seek (IN  KFileObject *   fileObj,
            IN  Int32           offset,
            IN  KFILE_FileSeek  origin) ;
```

Arguments

IN	KFileObject *	fileObj
	The fileObject to seek into.	
IN	Int32	offset
	Offset for positioning the file pointer.	
IN	KFILE_FileSeek	origin
	Origin for calculating absolute position where file pointer is to be positioned. This can take the following values: KFILE_SeekSet KFILE_SeekCur KFILE_SeekEnd	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid arguments.
DSP_EPOINTER	Invalid file object.
DSP_EFILE	File is not opened.
DSP_ERANGE	Offset and origin combination is beyond file size range.

Pre Conditions

Subcomponent must be initialized.

fileObj must be a valid handle to a file opened earlier.

Post Conditions

None

See Also

KFILE_Tell

29.8 KFILE_Tell

Returns the current file pointer position for the specified file handle.

Syntax

```
EXPORT_API
DSP_STATUS
KFILE_Tell (IN  KFileObject * fileObj,
            OUT Int32 *       pos) ;
```

Arguments

IN	KFileObject *	fileObj
	The fileObject pointer.	
OUT	Int32 *	pos
	Out argument for holding the current file position indicator value.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid arguments.
DSP_EPOINTER	Invalid file object.
DSP_EFILE	file is not opened.

Pre Conditions

Subcomponent must be initialized.

fileObj must be a valid handle to a file opened earlier.

Post Conditions

None

See Also

KFILE_Seek

30 mem.h

Defines the interfaces and data structures for the sub-component MEM.

Path

$(DSPLINK)\gpp\src\osal

Revision

00.08

CONSTANTS

30.1 MEM_DEFAULT

Default attributes for OS independent operations for memory allocation & de-allocation.

OS dependent attributes shall be defined in file 'mem_os.h'.

Syntax

```
#define MEM_DEFAULT    NULL
```

See Also

FUNCTIONS

30.2 MEM_Initialize

Initializes the MEM sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.

Pre Conditions

None.

Post Conditions

None.

See Also

None.

30.3 MEM_Finalize

Releases all resources used by this sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.

Pre Conditions

None.

Post Conditions

None.

See Also

None.

30.4 MEM_Alloc

Allocates the specified number of bytes.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Alloc (OUT Void ** ptr, IN Uint32 cBytes, IN OUT Pvoid arg) ;
```

Arguments

OUT	Void **	ptr
	Location where pointer to allocated memory will be kept .	
IN	Uint32	cBytes
	Number of bytes to allocate.	
IN OUT	Pvoid	arg
	Type of memory to allocate. MEM_DEFAULT should be used if there is no need of special memory.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

MEM must be initialized.

ptr must be a valid pointer.

cBytes must be greater than 0.

Post Conditions

*ptr must be a valid pointer upon successful completion otherwise

it must be NULL.

See Also

None

30.5 MEM_Calloc

Allocates the specified number of bytes and clears them by filling it with 0s.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Calloc (OUT Void ** ptr, IN Uint32 cBytes, IN OUT Pvoid arg) ;
```

Arguments

OUT	Void **	ptr
	Location where pointer to allocated memory will be kept	
IN	Uint32	cBytes
	Number of bytes to allocate.	
IN OUT	Pvoid	arg
	Type of memory to allocate. MEM_DEFAULT should be used if there is no need of special memory.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

MEM must be initialized.

ptr must be a valid pointer.

cBytes must be greater than 0.

Post Conditions

*ptr must be a valid pointer upon successful completion otherwise

it must be NULL.

See Also

None

30.6 MEM_Free

Frees up the allocated chunk of memory.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Free (IN Pvoid ptr, IN Pvoid arg) ;
```

Arguments

IN	Pvoid	ptr
	Pointer to start of memory to be freed.	
IN	Pvoid	arg
	Type of memory allocated.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

MEM must be initialized.

memBuf must be a valid pointer.

Post Conditions

None

@see

See Also

30.7 MEM_Map

Maps a specified memory area into the GPP virtual space.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Map (IN OUT MemMapInfo * mapInfo) ;
```

Arguments

IN OUT	MemMapInfo *	mapInfo
	Data required for creating the mapping.	

Return Values

DSP_SOK	Operation completed successfully.
DSP_EMEMORY	Could not map the given memory address.

Pre Conditions

mapInfo pointer must be valid.

Post Conditions

None

See Also

MEM_Unmap

30.8 MEM_Unmap

Unmaps the specified memory area.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Unmap (IN MemUnmapInfo * unmapInfo) ;
```

Arguments

IN	MemUnmapInfo *	unmapInfo
	Information required for unmapping a memory area.	

Return Values

DSP_SOK	Operation completed successfully.

Pre Conditions

unmapInfo pointer must be valid.

Post Conditions

None.

See Also

MEM_Map

30.9 MEM_Copy

Copies data between the specified memory areas.

Syntax

```
EXPORT_API
DSP_STATUS
MEM_Copy (IN Uint8 * dst, OUT Uint8 * src, IN Uint32 len, IN Endianism
endian) ;
```

Arguments

IN	Uint8 *	dst
	Destination address	
OUT	Uint8 *	src
	Source address	
IN	Uint32	len
	length of data to be coiped.	
IN	Endianism	endian
	Endianism	

Return Values

DSP_SOK	Operation completed successfully.

Pre Conditions

None.

Post Conditions

None.

See Also

None

30.10 MEM_Debug

Prints debug information for MEM.

Syntax

```
EXPORT_API
Void
MEM_Debug () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

31 mem_os.h

Defines the OS dependent attributes & structures for the sub-component MEM.

Path

$(DSPLINK)\gpp\src\osal\Linux

Revision

00.02

CONSTANTS

31.1 MEM_KERNEL

Indicates that memory must be allocated from kernel memory space.

Syntax

```
#define MEM_KERNEL      GFP_KERNEL
```

See Also

32 prcs.h

Defines the interfaces and data structures for the sub-component PRCS.

Path

`$(DSPLINK)\gpp\src\osal`

Revision

`00.06`

FUNCTIONS

32.1 PRCS_Initialize

Initializes the PRCS subcomponent.

Syntax

```
EXPORT_API
DSP_STATUS
PRCS_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

None

Post Conditions

None

See Also

None

32.2 PRCS_Finalize

Releases resources used by the PRCS subcomponent.

Syntax

```
EXPORT_API
DSP_STATUS
PRCS_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

None

Post Conditions

None

See Also

None

32.3 PRCS_Create

Creates a PrcsObject and populates it with information to identify the client.

Syntax

```
EXPORT_API
DSP_STATUS
PRCS_Create (OUT PrcsObject ** prcsObj) ;
```

Arguments

OUT	PrcsObject **	prcsObj
	OUT argument to store the created object.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

prcsObj must be a valid pointer.

Post Conditions

Valid object is returned in case of success.

See Also

PRCS_Delete

32.4 PRCS_Delete

Frees up resources used by the specified object.

Syntax

```
EXPORT_API
DSP_STATUS
PRCS_Delete (IN PrcsObject * prcsObj) ;
```

Arguments

IN	PrcsObject *	prcsObj
	Object to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid prcsObj

Pre Conditions

prcsObj must be a valid object.

Post Conditions

None

See Also

PRCS_Create

32.5 PRCS_IsEqual

Compares two clients to check if they are "equal". Equality is defined by implementation on the specific OS port.

Syntax

```
EXPORT_API
DSP_STATUS
PRCS_IsEqual (IN  PrcsObject *  client1,
              IN  PrcsObject *  client2,
              OUT Bool *        isEqual) ;
```

Arguments

IN	PrcsObject *	client1
	First client's information	
IN	PrcsObject *	client2
	Second client's information	
OUT	Bool *	isEqual
	Place holder for result of comparison	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

client1 must be a valid object.

client2 must be a valid object.

isEqual must be a valid pointer.

Post Conditions

None

See Also

PRCS_Create

32.6 PRCS_IsSameContext

Checks if the two clients share same context (address space).

Syntax

```
EXPORT_API
DSP_STATUS
PRCS_IsSameContext (IN  PrcsObject *  client1,
                    IN  PrcsObject *  client2,
                    OUT Bool *        isSame) ;
```

Arguments

IN	PrcsObject *	client1
	First client's information	
IN	PrcsObject *	client2
	Second client's information	
OUT	Bool *	isSame
	Place holder for result of comparison	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.

Pre Conditions

client1 must be a valid object.

client2 must be a valid object.

isSame must be a valid pointer.

Post Conditions

None

See Also

PRCS_Create

33 print.h

Interface declaration of OS printf abstraction.

Path

`$(DSPLINK)\gpp\src\osal`

Revision

`00.04`

FUNCTIONS

33.1 PRINT_Initialize

Initializes the PRINT sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
PRINT_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

None

Post Conditions

None

See Also

None

33.2 PRINT_Finalize

Releases resources used by this sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
PRINT_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.

Pre Conditions

None

Post Conditions

None

See Also

None

33.3 PRINT_Printf

Provides standard printf functionality abstraction.

Syntax

```
#if defined (TRACE_KERNEL)
EXPORT_API
Void
PRINT_Printf (Pstr format, ...) ;
```

Arguments

Pstr	format
Format string.	
	...
Variable argument list.	

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

34 sync.h

Defines the interfaces and data structures for the sub-component SYNC.

Path

$(DSPLINK)\gpp\src\osal

Revision

00.11

CONSTANTS

34.1 SYNC_WAITFOREVER

Argument used to wait forever in function SYNC_WaitOnEvent function.

Syntax

```
#define SYNC_WAITFOREVER            WAIT_FOREVER
```

See Also

34.2 SYNC_NOWAIT

Argument used for no waiting option in function SYNC_WaitOnEvent function.

Syntax

```
#define SYNC_NOWAIT                 WAIT_NONE
```

See Also

TYPE DEFINITIONS & STRUCTURES

34.3 SyncAttrs

This object contains various attributes of SYNC object.

Syntax

```
typedef struct SyncAttrs_tag {
    Uint32   flag ;
} SyncAttrs ;
```

Fields

Uint32	flag This flag is used by the various SYNC functions and its usage is dependent on the function using it.

See Also

None

34.4 SyncSemType

This enumeration defines the possible types of semaphores that can be created.

Syntax

```
typedef enum {
    SyncSemType_Binary   = 0,
    SyncSemType_Counting = 1
} SyncSemType ;
```

Fields

SyncSemType_Binary	Indicates that the semaphore is a binary semaphore.
SyncSemType_ Counting	Indicates that the semaphore is a counting semaphore.

See Also

None

FUNCTIONS

34.5 SYNC_Initialize

Initializes the SYNC subcomponent.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

None

Post Conditions

None

See Also

None

34.6 SYNC_Finalize

This function frees up all resources used by the SYNC subcomponent.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.

Pre Conditions

None

Post Conditions

None

See Also

None

34.7 SYNC_OpenEvent

Creates and initializes an event object for thread synchronization. The event is initialized to a non-signaled state.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_OpenEvent (OUT SyncEvObject ** event, IN SyncAttrs * attr) ;
```

Arguments

OUT	SyncEvObject **	event
	OUT argument to store the newly created event object.	
IN	SyncAttrs *	attr
	Reserved for future use.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EMEMORY	Operation failed due to insufficient memory.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

Pointer to event must be valid.

Pointer to attributes must be valid.

Post Conditions

Valid object is returned in case of success.

See Also

None

34.8 SYNC_CloseEvent

Closes the handle corresponding to an event. It also frees the resources allocated, if any, during call to SYNC_OpenEvent ().

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_CloseEvent (IN SyncEvObject * event) ;
```

Arguments

IN	SyncEvObject *	event
	Event to be closed.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

event must be a valid object.

Post Conditions

None

See Also

None

34.9 SYNC_ResetEvent

Resets the synchronization object to non-signaled state.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_ResetEvent (IN SyncEvObject * event) ;
```

Arguments

IN	SyncEvObject *	event
	Event to be reset.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

event must be a valid object.

Post Conditions

None

See Also

None

34.10 SYNC_SetEvent

Sets the state of synchronization object to signaled and unblocks all threads waiting for it.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_SetEvent (IN SyncEvObject * event) ;
```

Arguments

IN	SyncEvObject *	event
	Event to be set.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

event must be a valid object.

Post Conditions

None

See Also

None

34.11 SYNC_WaitOnEvent

Waits for an event to be signaled for a specified amount of time.

It is also possible to wait infinitely. This function must 'block' and not 'spin'.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_WaitOnEvent (IN SyncEvObject * event, IN Uint32 timeout) ;
```

Arguments

IN	SyncEvObject *	event
	Event to be waited on.	
IN	Uint32	timeout
	Timeout value.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_ETIMEOUT	Timeout occured while performing operation.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

event must be a valid object.

Post Conditions

None

See Also

None

34.12 SYNC_WaitOnMultipleEvents

Waits on multiple events. Returns when any of the event is set.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_WaitOnMultipleEvents (IN  SyncEvObject ** syncEvents,
                           IN  Uint32          count,
                           IN  Uint32          timeout,
                           OUT Uint32 *        index) ;
```

Arguments

IN	SyncEvObject **	syncEvents
	Array of events to be wait on.	
IN	Uint32	count
	Number of events.	
IN	Uint32	timeout
	Timeout for wait.	
OUT	Uint32 *	index
	OUT argument to store the index of event that was set.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_ETIMEOUT	Timeout occured while performing operation.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

syncEvents must be a valid object array.

index must be a valid pointer.

Post Conditions

None

See Also

None

34.13 SYNC_CreateCS

Initializes the Critical section structure.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_CreateCS (OUT SyncCsObject ** cSObj) ;
```

Arguments

OUT	SyncCsObject **	cSObj
	Structure to be intialized	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EMEMORY	Operation failed due to insufficient memory.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

cSObj must not be NULL.

Post Conditions

In case of success cSObj is valid.

See Also

None

34.14 SYNC_DeleteCS

Deletes the critical section object.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_DeleteCS (IN SyncCsObject * cSObj) ;
```

Arguments

IN	SyncCsObject *	cSObj
	Critical section to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

cSObj must be a valid object.

Post Conditions

None

See Also

None

34.15 SYNC_EnterCS

This function enters the critical section that is passed as an argument to it. After successful return of this function no other thread can enter until this thread exits the CS.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_EnterCS (IN SyncCsObject * cSObj) ;
```

Arguments

IN	SyncCsObject *	cSObj
	Critical section to enter.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

cSObj must be a valid object.

Post Conditions

None

See Also

None

34.16 SYNC_LeaveCS

This function makes the critical section available for other threads to enter.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_LeaveCS (IN SyncCsObject * cSObj) ;
```

Arguments

IN	SyncCsObject *	cSObj
	Critical section to leave.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EMEMORY	Operation failed due to insufficient memory.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

cSObj should be a valid object.

Post Conditions

None

See Also

None

34.17 SYNC_CreateSEM

Creates the semaphore object.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_CreateSEM (OUT SyncSemObject ** semObj, IN SyncAttrs * attr) ;
```

Arguments

OUT	SyncSemObject **	semObj
	Location to receive the pointer to the created semaphore object.	
IN	SyncAttrs *	attr
	Attributes to specify the kind of semaphore required to be created. For binary semaphores flag field in the attr should be set to SyncSemType_Binary. For counting semaphores flag field in the attr should be set to SyncSemType_Counting.	

Return Values

DSP_SOK	Semaphore object successfully created.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EINVALIDARG	Invalid arguments passed.
DSP_EMEMORY	Operation failed due to insufficient memory.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

semObj must not be NULL.

attr must not be NULL.

Post Conditions

In case of success semObj is valid.

See Also

None

34.18 SYNC_DeleteSEM

Deletes the semaphore object.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_DeleteSEM (IN SyncSemObject * semObj) ;
```

Arguments

IN	SyncSemObject *	semObj
	Pointer to semaphore object to be deleted.	

Return Values

DSP_SOK	Semaphore object successfully deleted.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

semObj must be a valid object.

Post Conditions

None

See Also

None

34.19 SYNC_WaitSEM

This function waits on the semaphore.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_WaitSEM (IN SyncSemObject * semObj, IN Uint32  timeout) ;
```

Arguments

IN	SyncSemObject *	semObj
	Pointer to semaphore object on which function will wait.	
IN	Uint32	timeout
	Timeout value.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_ETIMEOUT	Timeout occured while performing operation.
DSP_EPOINTER	Invalid pointer passed.

Pre Conditions

semObj must be a valid object.

Post Conditions

None

See Also

None

34.20 SYNC_SignalSEM

This function signals the semaphore and makes it available for other threads.

Syntax

```
EXPORT_API
DSP_STATUS
SYNC_SignalSEM (IN SyncSemObject * semObj) ;
```

Arguments

IN	SyncSemObject *	semObj
	Pointer to semaphore object to be signalled.	

Return Values

DSP_SOK	Operation successfully completed.
SYNC_E_FAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.
DSP_EMEMORY	Operation failed due to memory error.

Pre Conditions

semObj should be a valid object.

Post Conditions

None

See Also

None

34.21 SYNC_ProtectionStart

Marks the start of protected code execution.

Syntax

```
EXPORT_API
Void
SYNC_ProtectionStart () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

34.22 SYNC_ProtectionEnd

Marks the end of protected code execution.

Syntax

```
EXPORT_API
Void
SYNC_ProtectionEnd () ;
```

Arguments

None

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

34.23 SYNC_SpinLockStart

Begin protection of code through spin lock with all ISRs disabled.

Calling this API protects critical regions of code from preemption by tasks, DPCs and all interrupts.

This API can be called from DPC context.

Syntax

```
EXPORT_API
Uint32
SYNC_SpinLockStart () ;
```

Arguments

None

Return Values

ISR flags value to be passed to SYNC_SpinLockEnd.

Pre Conditions

None

Post Conditions

None

See Also

None

34.24 SYNC_SpinLockEnd

End protection of code through spin lock with all ISRs disabled.

This API can be called from DPC context.

Syntax

```
EXPORT_API
Void
SYNC_SpinLockEnd (IN Uint32 irqFlags) ;
```

Arguments

IN	Uint32	irqFlags	
	ISR flags value returned from SYNC_SpinLockStart ().		

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

35 trc.h

Defines the interfaces and data structures for the sub-component TRC.

Path

$(DSPLINK)\gpp\src\osal

Revision

00.07

CONSTANTS

35.1 MAXIMUM_COMPONENTS

maximum number of components supported.

Syntax

```
#define MAXIMUM_COMPONENTS          16
```

See Also

35.2 TRC_ENTER / TRC_LEVELn / TRC_LEAVE

Severity levels for debug printing.

Syntax

```
#define TRC_ENTER           0x01        /*  Lowest level of severity */
#define TRC_LEVEL1          0x02
#define TRC_LEVEL2          0x03
#define TRC_LEVEL3          0x04
#define TRC_LEVEL4          0x05
#define TRC_LEVEL5          0x06
#define TRC_LEVEL6          0x07
#define TRC_LEVEL7          0x08        /*  Highest level of severity */
#define TRC_LEAVE           TRC_ENTER
```

See Also

TYPE DEFINITIONS & STRUCTURES

35.3 TrcObject

TRC Object that stores the severity and component and subcomponent maps on a global level.

Syntax

```
typedef struct TrcObject_tag {
    Uint16 components ;
    Uint16 level      ;
    Uint16 subcomponents [MAXIMUM_COMPONENTS] ;
} TrcObject ;
```

Fields

Uint16	components component map
Uint16	level severity level
Uint16	subcomponents subcomponent map

See Also

FUNCTIONS

35.4 TRC_Enable

Enables debug prints on a component and subcomponent level.

Syntax

```
EXPORT_API
DSP_STATUS
TRC_Enable (IN Uint32 componentMap);
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent map	

Return Values

DSP_SOK	Operation successful
DSP_EINVALIDARG	Invalid argument to function call
DSP_EFAIL	Operation not successful

Pre Conditions

None

Post Conditions

None

See Also

TRC_Disable
TRC_SetSeverity

35.5 TRC_Disable

Disables debug prints on a component and subcomponent level.

Syntax

```
EXPORT_API
DSP_STATUS
TRC_Disable (IN Uint32 componentMap);
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent map	

Return Values

DSP_SOK	Operation successful
DSP_EINVALIDARG	Invalid argument to function call
DSP_EFAIL	Operation not successful

Pre Conditions

None

Post Conditions

None

See Also

TRC_Enable
TRC_SetSeverity

35.6 TRC_SetSeverity

set the severity of the required debug prints.

Syntax

```
EXPORT_API
DSP_STATUS
TRC_SetSeverity (IN Uint16   level) ;
```

Arguments

IN	Uint16	level
	The severity level of the debug prints required	

Return Values

DSP_SOK	Operation successful
DSP_EINVALIDARG	Invalid argument to function call
DSP_EFAIL	Operation not successful

Pre Conditions

None

Post Conditions

None

See Also

TRC_Enable
TRC_Disable

35.7 TRC_0Print

Prints a null terminated character string based on its severity, the subcomponent and component it is associated with.

Syntax

```
EXPORT_API
Void
TRC_0Print (IN  Uint32   componentMap,
            IN  Uint16   severity,
            IN  Char8 *  debugString) ;
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent to which this print belongs	
IN	Uint16	severity
	The severity associated with the print	
IN	Char8 *	debugString
	The null terminated character string to be printed	

Return Values

None

Pre Conditions

The character string is valid

Post Conditions

None

See Also

TRC_1Print
TRC_2Print
TRC_3Print
TRC_4Print
TRC_5Print
TRC_6Print

35.8 TRC_1Print

Prints a null terminated character string and an integer argument based on its severity, the subcomponent and component it is associated with.

Syntax

```
EXPORT_API
Void
TRC_1Print (IN  Uint32   componentMap,
            IN  Uint16   severity,
            IN  Char8 *  debugString,
            IN  Uint32   argument1) ;
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent to which this print belongs	
IN	Uint16	severity
	The severity associated with the print	
IN	Char8 *	debugString
	The null terminated character string to be printed	
IN	Uint32	argument1
	The integer argument to be printed	

Return Values

None

Pre Conditions

The character string is valid

Post Conditions

None

See Also

TRC_0Print
TRC_2Print
TRC_3Print
TRC_4Print
TRC_5Print
TRC_6Print

35.9 TRC_2Print

Prints a null terminated character string and two integer arguments based on its severity, the subcomponent and component it is associated with.

Syntax

```
EXPORT_API
Void
TRC_2Print (IN  Uint32   componentMap,
            IN  Uint16   severity,
            IN  Char8 *  debugString,
            IN  Uint32   argument1,
            IN  Uint32   argument2) ;
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent to which this print belongs	
IN	Uint16	severity
	The severity associated with the print	
IN	Char8 *	debugString
	The null terminated character string to be printed	
IN	Uint32	argument1
	The first integer argument to be printed	
IN	Uint32	argument2
	The second integer argument to be printed	

Return Values

None

Pre Conditions

The character string is valid

Post Conditions

None

See Also

TRC_0Print
TRC_1Print
TRC_3Print
TRC_4Print

```
    TRC_5Print
    TRC_6Print
```

35.10 TRC_3Print

Prints a null terminated character string and three integer arguments based on its severity, the subcomponent and component it is associated with.

Syntax

```
EXPORT_API
Void
TRC_3Print (IN  Uint32   componentMap,
            IN  Uint16   severity,
            IN  Char8 *  debugString,
            IN  Uint32   argument1,
            IN  Uint32   argument2,
            IN  Uint32   argument3) ;
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent to which this print belongs	
IN	Uint16	severity
	The severity associated with the print	
IN	Char8 *	debugString
	The null terminated character string to be printed	
IN	Uint32	argument1
	The first integer argument to be printed	
IN	Uint32	argument2
	The second integer argument to be printed	
IN	Uint32	argument3
	The third integer argument to be printed	

Return Values

None

Pre Conditions

The character string is valid

Post Conditions

None

See Also

TRC_0Print
TRC_1Print
TRC_2Print
TRC_4Print
TRC_5Print
TRC_6Print

35.11 TRC_4Print

Prints a null terminated character string and four integer arguments based on its severity, the subcomponent and component it is associated with.

Syntax

```
EXPORT_API
Void
TRC_4Print (IN  Uint32   componentMap,
            IN  Uint16   severity,
            IN  Char8 *  debugString,
            IN  Uint32   argument1,
            IN  Uint32   argument2,
            IN  Uint32   argument3,
            IN  Uint32   argument4) ;
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent to which this print belongs	
IN	Uint16	severity
	The severity associated with the print	
IN	Char8 *	debugString
	The null terminated character string to be printed	
IN	Uint32	argument1
	The first integer argument to be printed	
IN	Uint32	argument2
	The second integer argument to be printed	
IN	Uint32	argument3
	The third integer argument to be printed	
IN	Uint32	argument4
	The fourth integer argument to be printed	

Return Values

None

Pre Conditions

The character string is valid

Post Conditions

None

See Also

TRC_0Print
TRC_1Print
TRC_2Print
TRC_3Print
TRC_5Print
TRC_6Print

35.12 TRC_5Print

Prints a null terminated character string and five integer arguments based on its severity, the subcomponent and component it is associated with.

Syntax

```
EXPORT_API
Void
TRC_5Print (IN  Uint32   componentMap,
            IN  Uint16   severity,
            IN  Char8 *  debugString,
            IN  Uint32   argument1,
            IN  Uint32   argument2,
            IN  Uint32   argument3,
            IN  Uint32   argument4,
            IN  Uint32   argument5) ;
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent to which this print belongs	
IN	Uint16	severity
	The severity associated with the print	
IN	Char8 *	debugString
	The null terminated character string to be printed	
IN	Uint32	argument1
	The first integer argument to be printed	
IN	Uint32	argument2
	The second integer argument to be printed	
IN	Uint32	argument3
	The third integer argument to be printed	
IN	Uint32	argument4
	The fourth integer argument to be printed	
IN	Uint32	argument5
	The fifth integer argument to be printed	

Return Values

None

Pre Conditions

The character string is valid

Post Conditions

None

See Also

TRC_0Print
TRC_1Print
TRC_2Print
TRC_3Print
TRC_4Print
TRC_6Print

35.13 TRC_6Print

Prints a null terminated character string and six integer arguments based on its severity, the subcomponent and component it is associated with.

Syntax

```
EXPORT_API
Void
TRC_6Print (IN  Uint32   componentMap,
            IN  Uint16   severity,
            IN  Char8 *  debugString,
            IN  Uint32   argument1,
            IN  Uint32   argument2,
            IN  Uint32   argument3,
            IN  Uint32   argument4,
            IN  Uint32   argument5,
            IN  Uint32   argument6) ;
```

Arguments

IN	Uint32	componentMap
	The component & subcomponent to which this print belongs	
IN	Uint16	severity
	The severity associated with the print	
IN	Char8 *	debugString
	The null terminated character string to be printed	
IN	Uint32	argument1
	The first integer argument to be printed	
IN	Uint32	argument2
	The second integer argument to be printed	
IN	Uint32	argument3
	The third integer argument to be printed	
IN	Uint32	argument4
	The fourth integer argument to be printed	
IN	Uint32	argument5
	The fifth integer argument to be printed	
IN	Uint32	argument6

	The sixth integer argument to be printed

Return Values

None

Pre Conditions

The character string is valid

Post Conditions

None

See Also

TRC_0Print
TRC_1Print
TRC_2Print
TRC_3Print
TRC_4Print
TRC_5Print.

36 drv_api.h

User side driver wrapper interface.

Path

`$(DSPLINK)\gpp\src\api\Linux`

Revision

`00.03`

37 drv_pmgr.h

Linux module driver interface file.

Path

$(DSPLINK)\gpp\src\pmgr\Linux

Revision

00.08

TYPE DEFINITIONS & STRUCTURES

37.1 CMD_Args

Union defining arguments to be passed to ioctl calls. For the explanation of individual field please see the corresponding APIs.

Syntax

```
typedef struct CMD_Args_tag {
      /* union of various arguments */
} ;
```

Fields

See Also

37.2 DrvAddrMapEntry

Structure for the entry of user-kernel address mapping table.

Syntax

```
typedef struct DrvAddrMapEntry_tag {
    Void * userAddress     ;
    Void * kernelAddress   ;
    Void * physicalAddress ;
    Uint32 length          ;
    Bool   valid           ;
} DrvAddrMapEntry ;
```

Fields

Void *	userAddress User side address of the buffer.
Void *	kernelAddress kernel side address of the buffer.
Void *	physicalAddress Physical address of the buffer.
Uint32	length Total length of the mapped memory chunk.
Bool	valid Tells if the entry is valid or not.

See Also

GENERIC HELPER FUNCTIONS

38 list.h

Declarations of list management control structures and definitions of inline list management functions.

Path

$(DSPLINK)\gpp\src\gen

Revision

00.05

TYPE DEFINITIONS & STRUCTURES

38.1 ListElement

An element of a list.

Syntax

```
typedef struct ListElement_tag {
    struct ListElement_tag *  next ;
    struct ListElement_tag *  prev ;
} ListElement ;
```

Fields

struct ListElement_tag *	next Next node pointer.
struct ListElement_tag *	prev Previous node pointer.

See Also

38.2 List

Definition of a List.

Syntax

```
typedef struct List_tag {
    ListElement    head ;
} List ;
```

Fields

ListElement	head
	Head of the list.

See Also

FUNCTIONS

38.3 LIST_Initialize

Initializes private state of LIST sub-component.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

None.

Post Conditions

LIST is initialized.

See Also

LIST_Finalize

38.4 LIST_Finalize

Discontinues usage of module.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be Initialized.

Post Conditions

Resources used by module are freed when reference count reaches zero.

See Also

LIST_Initialize

38.5 LIST_Create

Allocates and initializes a circular list.

Uses portable MEM_Calloc () function to allocate a list containing a single element and initializes that element to indicate that it is the "end of the list" (i.e., the list is empty).

An empty list is indicated by the "next" pointer in the element at the head of the list pointing to the head of the list, itself.

The created list contains a single element. This element is the "empty" element, because its "next" and "prev" pointers point at the same location (the element itself).

Syntax

```
EXPORT_API
DSP_STATUS
LIST_Create (OUT List ** list) ;
```

Arguments

OUT	List **	list
	OUT parameter containing the created list.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EOUTOFMEMORY	Out of memory.

Pre Conditions

LIST must be initialized.

Post Conditions

None

See Also

None

38.6 LIST_Delete

Removes a list by freeing its control structure's memory space.

Uses portable MEM_Free () function to deallocate the memory block pointed at by the input parameter.

Must ONLY be used for empty lists, because it does not walk the chain of list elements. Calling this function on a non-empty list will cause a memory leak.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_Delete (IN List * list) ;
```

Arguments

IN	List *	list
	Pointer to list control structure of list to be deleted.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be initialized.

List must be valid.

Post Conditions

None

See Also

LIST_Create

38.7 LIST_InitializeElement

Initializes a list element to default (cleared) values.

This function must not be called to "reset" an element in the middle of a list chain -- that would break the chain.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_InitializeElement (IN ListElement * element) ;
```

Arguments

IN	ListElement *	element
	Pointer to list element to be reset.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be initialized.

Post Conditions

None

See Also

LIST_InsertBefore
LIST_PutTail

38.8 LIST_InsertBefore

Inserts the element before the existing element.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_InsertBefore (IN  List *          list,
                   IN  ListElement *   insertElement,
                   IN  ListElement *   existingElement) ;
```

Arguments

IN	List *	list
	Pointer to list control structure.	
IN	ListElement *	insertElement
	Pointer to element in list to insert.	
IN	ListElement *	existingElement
	Pointer to existing list element.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be initialized.

list must be valid.

pInsertElement must be valid.

pExistingElement must be valid.

Post Conditions

None

See Also

LIST_Create
LIST_PutTail

38.9 LIST_PutTail

Adds the specified element to the tail of the list.

Sets new element's "prev" pointer to the address previously held by the head element's prev pointer. This is the previous tail member of the list.

Sets the new head's prev pointer to the address of the element.

Sets next pointer of the previous tail member of the list to point to the new element (rather than the head, which it had been pointing at).

Sets new element's next pointer to the address of the head element.

Sets head's prev pointer to the address of the new element.

Because the tail is always "just before" the head of the list (the tail's "next" pointer points at the head of the list, and the head's "prev" pointer points at the tail of the list), the list is circular.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_PutTail (IN List * list, IN ListElement * element) ;
```

Arguments

IN	List *	list
	Pointer to list control structure to which *pElem will be added.	
IN	ListElement *	element
	Pointer to list element to be added.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

*element and *list must both exist.

LIST must be initialized.

Post Conditions

None

See Also

LIST_InsertBefore

38.10 LIST_RemoveElement

Removes (unlinks) the given element from the list, if the list is not empty. Does not free the list element.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_RemoveElement (IN List * list, IN ListElement * element) ;
```

Arguments

IN	List *	list
	Pointer to list control structure.	
IN	ListElement *	element
	Pointer to element in list to remove.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be initialized.

list must be valid.

element must be valid.

Post Conditions

None

See Also

```
LIST_InsertBefore
LIST_Create
LIST_PutTail
```

38.11 LIST_First

Returns a pointer to the first element of the list, or NULL if the list is empty.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_First (IN List * list, OUT ListElement ** element) ;
```

Arguments

IN	List *	list
	Pointer to list control structure.	
OUT	ListElement **	element
	OUT parameter for holding the first element.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be initialized.

list must be valid.

Post Conditions

None

See Also

```
LIST_Create
LIST_PutTail
LIST_InsertBefore
```

38.12 LIST_GetHead

Pops the head off the list and returns a pointer to it.

If the list is empty, returns NULL.

Else, removes the element at the head of the list, making the next element the head of the list.

The head is removed by making the tail element of the list point its "next" pointer at the next element after the head, and by making the "prev" pointer of the next element after the head point at the tail element. So the next element after the head becomes the new head of the list.

Because the tail of the list points forward (its "next" pointer) to the head of the list, and the head of the list points backward (its "prev" pointer) to the tail of the list, this list is circular.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_GetHead (IN List * list, OUT ListElement ** pHeadElement) ;
```

Arguments

IN	List *	list
	Pointer to list control structure of list whose head. element is to be removed.	
OUT	ListElement **	pHeadElement
	OUT Parameter to hold the head element.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be initialized.

list must be valid.

pHeadElement must be valid.

Post Conditions

None

See Also

LIST_PutTail
LIST_InsertBefore

38.13 LIST_Next

Returns a pointer to the next element of the list, or NULL if the next element is the head of the list or the list is empty.

Syntax

```
EXPORT_API
DSP_STATUS
LIST_Next (IN  List *          list,
           IN  ListElement *   pCurrentElement,
           OUT ListElement **  pNextElement) ;
```

Arguments

IN	List *	list
	Pointer to list control structure.	
IN	ListElement *	pCurrentElement
	Pointer to element in list to remove.	
OUT	ListElement **	pNextElement
	OUT parameter to hold the next element.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

LIST must be initialized.

list must be valid.

pCurrentElem must be valid.

Post Conditions

None

See Also

LIST_InsertBefore
LIST_PutTail

39 buf.h

Defines the interfaces and data structures for the BUF sub-component.

Path

$(DSPLINK)\gpp\src\gen

Revision

00.03

TYPE DEFINITIONS & STRUCTURES

39.1 BufObj

This structure defines the buffer pool object. It maintains the pool of buffers of a particular fixed size.

Syntax

```
typedef struct BufObj_tag {
    Uint32        startAddress ;
    Uint16        size         ;
    Uint32        nextFree     ;
    Uint16        totalBuffers ;
    Uint16        freeBuffers  ;
    Bool          freePool     ;
} BufObj ;
```

Fields

Uint32	startAddress Starting address of buffer pool.
Uint16	size Size of the buffers in this pool.
Uint32	nextFree Pointer to next free buffer.
Uint16	totalBuffers Total number of buffers in pool.
Uint16	freeBuffers Number of free buffers in pool.
Bool	freePool Indicates whether the buffer pool was allocated within the BUF component, and should be freed during BUF_Delete ()

See Also

CONSTANTS

39.2 MIN_BUF_SIZE

This constant defines the minimum size of the buffer to be allocated.

Syntax

```
#define MIN_BUF_SIZE (sizeof (BufHeader))
```

See Also

FUNCTIONS

39.3 BUF_Initialize

This function initializes the buffer component.

Syntax

```
EXPORT_API
DSP_STATUS
BUF_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

None

Post Conditions

None

See Also

BUF_Finalize

39.4 BUF_Finalize

This function finalizes the buffer component.

Syntax

```
EXPORT_API
DSP_STATUS
BUF_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General failure.

Pre Conditions

None

Post Conditions

None

See Also

BUF_Initialize

39.5 BUF_Create

This function creates and initializes a fixed buffer pool and returns a handle to the corresponding buffer pool object.

Syntax

```
EXPORT_API
DSP_STATUS
BUF_Create (IN     Uint16       numBufs,
            IN     Uint16       size,
            OUT    BufHandle  * bufHandle,
            IN OPT Uint32       bufAddress) ;
```

Arguments

IN	Uint16	numBufs
	Number of buffers to be created in the pool.	
IN	Uint16	size
	Size of the buffers within the pool.	
OUT	BufHandle *	bufHandle
	Location to receive the handle to the created buffer pool object.	
IN OPT	Uint32	bufAddress
	Address of the memory reserved for this buffer pool. If a valid address is specified, no memory is allocated within this function, and it can be called from within DPC context. The size of the memory for the buffer pool allocated by the user must be equal to (size * numBufs).If the address specified is NULL, this function internally allocates the memory required for the buffer pool. In this case, this function cannot be called from DPC or ISR context.	

Return Values

DSP_SOK	The buffer pool has been successfully created.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

bufHandle must be valid.

size must be greater than or equal to MIN_BUF_SIZE.

numBufs must be greater than 0.

Post Conditions

None

See Also

BufHeader
BufObj
BUF_Delete

39.6 BUF_Delete

This function deletes the buffer pool specified by the user.

Syntax

```
EXPORT_API
DSP_STATUS
BUF_Delete (IN  BufHandle  bufHandle) ;
```

Arguments

IN	BufHandle	bufHandle
	Handle to the buffer pool object.	

Return Values

DSP_SOK	The buffer pool has been successfully deleted.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

bufHandle must be valid.

Post Conditions

None

See Also

BufObj
BUF_Create

39.7 BUF_Alloc

This function allocates a free buffer from the specified buffer pool and returns it to the user.

Syntax

```
EXPORT_API
DSP_STATUS
BUF_Alloc (IN  BufHandle  bufHandle, OUT Pvoid * buffer) ;
```

Arguments

IN	BufHandle	bufHandle
	Handle to the buffer pool object.	
OUT	Pvoid *	buffer
	Location to receive the allocated buffer.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

bufHandle must be valid.

buffer must be valid.

Post Conditions

None

See Also

BufObj
BUF_Free

39.8 BUF_Free

This function frees the buffer specified by the user, and returns it to the buffer pool.

Syntax

```
EXPORT_API
DSP_STATUS
BUF_Free (IN  BufHandle  bufHandle, IN  Pvoid  buffer) ;
```

Arguments

IN	BufHandle	bufHandle
	Handle to the buffer pool object.	
IN	Pvoid	buffer
	Pointer to the buffer to be freed.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

bufHandle must be valid.

buffer must be valid.

Post Conditions

None

See Also

BufObj
BUF_Alloc

39.9 BUF_GetStats

This function gets instrumentation information about the specified buffer pool.

Syntax

```
EXPORT_API
DSP_STATUS
BUF_GetStats (IN  BufHandle  bufHandle, OUT BufStats * bufStats) ;
```

Arguments

IN	BufHandle	bufHandle
	Handle to the buffer pool object.	
OUT	BufStats *	bufStats
	Location to receive the instrumentation information.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.
DSP_EFAIL	General failure.

Pre Conditions

bufHandle must be valid.

bufStats must be valid.

Post Conditions

None

See Also

BufObj
BUF_Alloc

40 gen_utils.h

Platform independent common library function interface.

Path

$(DSPLINK)\gpp\src\gen

Revision

00.05

TYPE DEFINITIONS & STRUCTURES

40.1 ErrorInfo

Structure for storing error reason.

Syntax

```
typedef struct ErrorInfo_tag {
    Bool       IsSet    ;

    DSP_STATUS ErrCode  ;

    Int32      OsMajor  ;
    Int32      OsMinor  ;
    Int32      OsBuild  ;

    Int32      PddMajor ;
    Int32      PddMinor ;
    Int32      PddBuild ;

    Int32      FileId   ;
    Int32      LineNum  ;
} ErrorInfo ;
```

Fields

Bool	IsSet Flag to indicate error is set.
DSP_STATUS	ErrCode Error Code.
Int32	OsMajor OS Version Major version number.
Int32	OsMinor OS Version Minor version number.
Int32	OsBuild OS Version Build number.

Int32	PddMajor PDD Version Major version number.
Int32	PddMinor PDD Version Minor version number.
Int32	PddBuild PDD Version Build number.
Int32	FileId ID of the file where failure occured.
Int32	LineNum Line number where failure occured.

See Also

FUNCTIONS

40.2 GEN_Initialize

Initializes the GEN module's private state.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_Initialize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

None

Post Conditions

Subcomponent is initialized.

See Also

GEN_Finalize

40.3 GEN_Finalize

Discontinues usage of the GEN module.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_Finalize () ;
```

Arguments

None

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.

Pre Conditions

Subcomponent must be initialized.

Post Conditions

None

See Also

GEN_Initialize

40.4 GEN_SetReason

This function logs failure if no previous failure has been logged.

Syntax

```
EXPORT_API
Void
GEN_SetReason (DSP_STATUS status, Int32 FileId, Int32 Line) ;
```

Arguments

DSP_STATUS	status
Error status to be logged.	
Int32	FileId
File identifier.	
Int32	Line
Line number where error occurred.	

Return Values

None

Pre Conditions

None

Post Conditions

None

See Also

None

40.5 GEN_NumToAscii

Converts a 1 or 2 digit number to a 2 digit string.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_NumToAscii (IN Uint32 number, OUT Char8 * strNumber) ;
```

Arguments

IN	Uint32	number
	Number to convert.	
OUT	Char8 *	strNumber
	Buffer to store converted string.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.

Pre Conditions

Subcomponent must be initialized.

The number to convert must be between 0 and 99, both numbers included.

The buffer to store output string must be valid.

Post Conditions

The buffer to store output string is valid.

See Also

None

40.6 GEN_Strcmp

Compares 2 ASCII strings. Works the same way as stdio's strcmp.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_Strcmp (IN  CONST Char8 * string1,
            IN  CONST Char8 * string2,
            OUT       Int32 * cmpResult) ;
```

Arguments

IN CONST	Char8 *	string1
	First string for comparison.	
IN CONST	Char8 *	string2
	Second string for comparison.	
OUT	Int32 *	cmpResult
	Result of comparision (zero = equal, non-zero otherwise).	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.

Pre Conditions

Subcomponent must be initialized.

The buffer to store first string must be valid.

The buffer to store second string must be valid.

Post Conditions

None

See Also

None

40.7 GEN_Strcpyn

Safe strcpy function.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_Strcpyn (OUT Char8 * destination,
             IN  Char8 * source,
             IN  Int32   maxNum) ;
```

Arguments

OUT	Char8 *	destination
	destination buffer.	
IN	Char8 *	source
	Source buffer.	
IN	Int32	maxNum
	Number of characters to copy.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.

Pre Conditions

Subcomponent must be initialized.

The destination buffer must be valid.

The source buffer must be valid.

The number of characters to copy must be more than zero.

Post Conditions

None

See Also

None

40.8 GEN_Strlen

Determines the length of a null terminated ASCI string.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_Strlen (IN CONST Char8 * str, OUT Uint32 * length) ;
```

Arguments

IN CONST	Char8 *	str
	Pointer to string.	
OUT	Uint32 *	length
	Out parameter to hold the length of string.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.

Pre Conditions

Subcomponent must be initialized.

The pointer to the string buffer must be valid.

The pointer to the length field must be valid.

Post Conditions

The pointer to the length field is valid.

See Also

None

40.9 GEN_WcharToAnsi

Converts a wide char string to an ansi string.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_WcharToAnsi (OUT Char8  * dest,
                 IN  Char16 * source,
                 IN  Int32    numChars) ;
```

Arguments

OUT	Char8 *	dest
	Destination buffer.	
IN	Char16 *	source
	Source buffer.	
IN	Int32	numChars
	Number of characters (Wide chars) to be converted/copied.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.

Pre Conditions

Subcomponent must be initialized.

The destination buffer must be valid.

The source buffer must be valid.

The number of characters to be converted/copied must be greater than 0.

Post Conditions

The destination buffer is valid.

See Also

None

40.10 GEN_Wstrlen

Determines the length of a null terminated wide character string.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_Wstrlen (IN Char16 * str, IN Uint32 * length) ;
```

Arguments

IN	Char16 *	str
	Pointer to string.	
IN	Uint32 *	length
	Length	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.

Pre Conditions

Subcomponent must be initialized.

The pointer to the string buffer must be valid.

The pointer to length of buffer must be valid.

Post Conditions

The pointer to length of buffer is valid.

See Also

None

40.11 GEN_Strcatn

Safe strcat function.

Syntax

```
EXPORT_API
DSP_STATUS
GEN_Strcatn (OUT Char8 * destination,
             IN  Char8 * source,
             IN  Int32   maxNum) ;
```

Arguments

OUT	Char8 *	destination
	destination buffer.	
IN	Char8 *	source
	Source buffer.	
IN	Int32	maxNum
	Maximum length of the destination buffer after concatenation.	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.

Pre Conditions

The destination buffer must be valid.

The source buffer must be valid.

The number of characters to copy must be more than zero.

Post Conditions

None.

See Also

None

41 coff.h

Defines COFF loader interface.

Path

$(DSPLINK)\gpp\src\gen

Revision

00.09

CONSTANTS

41.1 SWAP_LOCATION

Location in COFF file where swap information is kept.

Syntax

```
#define     SWAP_LOCATION                   20
```

See Also

41.2 SECT_DSECT

Identifier for dummy section.

Syntax

```
#define     SECT_DSECT                      0x0001
```

See Also

41.3 SECT_NOLOAD

Identifier for a no_load section.

Syntax

```
#define     SECT_NOLOAD                     0x0002
```

See Also

41.4 SECT_BSS

Identifier for a BSS section.

Syntax

```
#define     SECT_BSS                        0x0080
```

See Also

41.5 SECT_COPY

Identifier for a COPY section.

Syntax

```
#define     SECT_COPY                       0x0010
```

See Also

41.6 COFF_VERSION

Offset in file header where number of sections is present.

Syntax

```
#define     COFF_VERSION                        0x00C2
```

See Also

41.7 SIZE_OPT_HDR_LOC

Location in file header for number of bytes in optional header.

Syntax

```
#define     SIZE_OPT_HDR_LOC     16
```

See Also

41.8 COFF_NAME_LEN

Length of name.

Syntax

```
#define     COFF_NAME_LEN                       8
```

See Also

41.9 SYMTAB_OFFSET

Offset in file header where symbol table details are present.

Syntax

```
#define     SYMTAB_OFFSET                       8
```

See Also

41.10 NUM_SECT_OFFSET

Offset in file header where number of sections is present.

Syntax

```
#define     NUM_SECT_OFFSET                     2
```

See Also

41.11 SIZE_COFF_FILE_HEADER/SIZE_COFF_SYMBOL_ENTRY/

Size of file header, symbolEntry and sectionHeader structure in COFF file format.

Syntax

```
#define     SIZE_COFF_FILE_HEADER             22#define
SIZE_COFF_SYMBOL_ENTRY            18#define
SIZE_COFF_SECTION_HEADER          48
```

See Also

TYPE DEFINITIONS & STRUCTURES

41.12 CoffFileHeader

File header for a COFF file.

Syntax

```
typedef struct CoffFileHeader_tag {
    Uint16  version          ;
    Uint16  numSections      ;
    Int32   dateTime         ;
    Int32   fpSymTab         ;
    Int32   numSymTabEntries ;
    Uint16  numBytesOptHeader ;
    Uint16  flags            ;
    Uint16  targetId         ;
} CoffFileHeader ;
```

Fields

Uint16	version Version ID. indicates the version of the COFF file structure.
Uint16	numSections Number of section headers
Int32	dateTime Time and date stamp. indicates when the file was created.
Int32	fpSymTab Symbol table's starting location in file.
Int32	numSymTabEntries Number of entries in the symbol table.
Uint16	numBytesOptHeader Number of bytes in the optional header. This field is either 0 or 28. If it is 0, there is no optional file header.
Uint16	flags Flags (see the File Header Flags table).
Uint16	targetId Target ID. magic number indicates the file can be

executed in a particular system. This field is checked for validating the support of supplied file.

See Also

41.13 CoffContext

Structure defining the context of loader. This object is created on initialization of this sub component and it is required to be passed as a parameter for any subsequent function call.

Syntax

```
typedef struct CoffContext_tag {
    KFileObject * fileObj    ;
    Uint32        startAddr  ;
    Bool          isSwapped  ;
} CoffContext ;
```

Fields

KFileObject *	fileObj File object for the DSP base image file.
Uint32	startAddr Entry point address for the DSP base image file.
Bool	isSwapped Flag to indicate if the file data is swapped.

See Also

41.14 CoffOptHeader

Optional header for coff file format.

Syntax

```
typedef struct CoffOptHeader_tag {
    Int16 magic          ;
    Int16 version        ;
    Int32 sizeExeCode    ;
    Int32 sizeInitData   ;
    Int32 sizeUninitData ;
    Int32 entry          ;
    Int32 addrExe        ;
    Int32 addrInitData   ;
} CoffOptHeader ;
```

Fields

Int16	magic Optional file header magic number
Int16	version Version stamp.
Int32	sizeExeCode Size (in bytes) of executable code.
Int32	sizeInitData Size (in bytes) of initialized data.
Int32	sizeUninitData Size (in bytes) of uninitialized data.
Int32	entry Entry point.
Int32	addrExe Beginning address of executable code.
Int32	addrInitData Beginning address of initialized data.

See Also

41.15 CoffSectionHeader

Section header for COFF file format.

Syntax

```
typedef struct CoffSectionHeader_tag {
    Char8   name [COFF_NAME_LEN] ;
    Int32   physicalAddress      ;
    Int32   virtualAddress       ;
    Int32   size                 ;
    Int32   fpRawData            ;
    Int32   fpReloc              ;
    Int32   fpLineNum            ;
    Uint32  numReloc             ;
    Uint32  numLine              ;
    Uint32  flags                ;
    Uint16  reserved             ;
    Uint16  memPageNum           ;
    Bool    isLoadSection        ;
    Char8 * data                 ;
} CoffSectionHeader ;
```

Fields

Char8	name This field contains one of the following: 1) An 8-character section name, padded with nulls, or 2) A pointer into the string table if the section name is longer than 8 characters. In the latter case the first four bytes of the field are 0.
Int32	physicalAddress Section's physical address.
Int32	virtualAddress Section's virtual address.
Int32	size Section's size in bytes.
Int32	fpRawData File pointer to raw data.
Int32	fpReloc File pointer to relocation entries.
Int32	fpLineNum

	File pointer to line-number entries.
Uint32	numReloc Number of relocation entries.
Uint32	numLine Number of line-number entries.
Uint32	flags Flags (see the Section Header Flags table)
Uint16	reserved Reserved.
Uint16	memPageNum Memory page number.
Bool	isLoadSection Flag to indicate that the section is loadable.
Char8 *	data Buffer to hold data.

See Also

41.16 CoffSymbolEntry

Defines the structure for a symbol table entry.

Syntax

```
typedef struct CoffSymbolEntry_tag {
    Char8  name [COFF_NAME_LEN] ;
    Int32  value                ;
    Int16  sectNum              ;
    Uint16 type                 ;
    Char8  storage              ;
    Char8  numAuxEnt            ;
} CoffSymbolEntry ;
```

Fields

Char8	name This field contains one of the following: 1) An 8-character symbol name, padded with nulls. 2) A pointer into the string table if the symbol name is longer than 8 characters. In the later case the first four bytes of the field are 0.
Int32	value Symbol value; storage class dependent.
Int16	sectNum Section number of the symbol.
Uint16	type Basic and derived type specification.
Char8	storage Storage class of the symbol.
Char8	numAuxEnt Number of auxiliary entries (always 0 or 1). If this is '1' then this structure is followed by the Auxilliary entry structure (which is of the same size as this structure).

See Also

42 coff_int.h

Defines interface for generic functions and macros of COFF loader.

Path

$(DSPLINK)\gpp\src\gen

Revision

00.01

FUNCTIONS

42.1 COFF_Read8

Reads an Int8 from file.

Syntax

```
NORMAL_API
Int8
COFF_Read8 (IN KFileObject * fileObj) ;
```

Arguments

IN	KFileObject *	fileObj
	File to read from.	

Return Values

The read value.

Pre Conditions

fileObj must be valid.

Post Conditions

None

See Also

None

42.2 COFF_Read16

Reads an Int16 from file.

Syntax

```
NORMAL_API
Int16
COFF_Read16 (IN KFileObject * fileObj, IN Bool swap) ;
```

Arguments

IN	KFileObject *	fileObj
	File to read from.	
IN	Bool	swap
	specifies whether the bytes need to be swapped.	

Return Values

The read value.

Pre Conditions

fileObj must be valid.

Post Conditions

None

See Also

None

42.3 COFF_Read32

Reads an Int32 from file.

Syntax

```
NORMAL_API
Int32
COFF_Read32 (IN KFileObject * fileObj, IN Bool swap) ;
```

Arguments

IN	KFileObject *	fileObj
	File to read from.	
IN	Bool	swap
	specifies whether the bytes need to be swapped.	

Return Values

The read value.

Pre Conditions

fileObj must be valid.

Post Conditions

None

See Also

None

<End Of Document>

« « « § » » »