DSP/BIOS™ LINK

PROCESSOR MANAGER

LNK 010 DES

Version 1.11

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

1 Introduction....6
1.1 Purpose and Scope....6
1.2 Terms and Abbreviations....6
1.3 References....6
1.4 Overview....7
2 High Level Design....8
3 PMGR_PROC....10
3.1 Resources Available....10
3.2 Dependencies....10
3.3 Description....10
3.4 Typedefs and Data Structures....11
3.5 API Definition....13
4 PMGR_CHNL....23
4.1 Resources Available....23
4.2 Description....23
4.3 API Definition....24
5 PMGR_CODE....34
5.1 Description....34
5.2 API Definition....35
6 PMGR_PARS....38
6.1 Resources Available....38
6.2 Description....38
6.3 Typedefs and Data Structures....39
6.4 API Definition....42
7 Appendix....50
7.1 Concept of Ownership of Components....50
7.2 Future Enhancements....50

TABLE OF FIGURES

Figure 1. Relationship Between the Components in Processor Manager and DSP/BIOS™ Link.... 8
Figure 2. Components involved in parsing a DSP executable....34

1 Introduction

1.1 Purpose and Scope

This document describes the overall design and architecture of the Processor Manager layer of the DSP/BIOS™ Link. The initial implementation of Processor Manager is intended for the DSP/BIOS™ LINK on the OMAP running Nucleus.

It lists the interfaces that the PMGR layer exposes and also describes the overall design for implementing these interfaces.

Return values as returned by a function in the document may not reflect all possible values that the function returns.

1.2 Terms and Abbreviations

CFG	Configuration sub-component
PMGR_CHNL	Channel sub-component
COFF	Common Object File Format
GPP	General Purpose Processor
LDRV	Link Driver sub-component
LIST	A collection of methods that allow list management.
OMAP	TI's multicore chipset
PGMR	Processor Manager component
PMGR_PARS	Parser sub-component
PMGR_PROC	Processor sub-component
User API	Application Programming Interface exposed by DSP/BIOS™ LINK

1.3 References

1.	LNK 001 PRD	DSP/BIOS™ LINK Product Requirements Document Version 1.00, dated JUN 12, 2002
2.	LNK 002 ARC	DSP/BIOS™ LINK High Level Architecture Version 1.02, dated JUL 15, 2003
3	LNK 012 DES	DSP/BIOS™ LINK Link Driver Version 1.11, dated JUL 25, 2003

1.4 Overview

The Processor Manager forms the layer of DSP/BIOS™ Link that is exported to the user. It provides functionality to both, control the DSP i.e., load code, start the DSP image execution, stop it etc., and transfer the data through the data streams or channels between the GPP and the DSP. The Processor Manager is also responsible for parsing the image file before loading it onto the DSP. It uses the services of the Link Driver to perform the tasks for a user.

The Processor Manager's individual subcomponents implement this policy:

a. The first client that starts using a resource (PMGR_PROC/PMGR_CHNL) is designated as the owner of the resource.

b. It frees the resource only when the owner releases it.

If the owner frees a resource, the resource is released even if the other clients have not yet released the resource. In such a case, the other clients (if any) are notified about the release of the resource.

2 High Level Design

The Processor Manager implements its dual functionality of control and communication with the DSP, using services from the Link Driver and the GPP OS services from the OSAL.

Figure 1 shows the relationship of components in the Processor Manager layer with other components of DSP/BIOS™ Link.

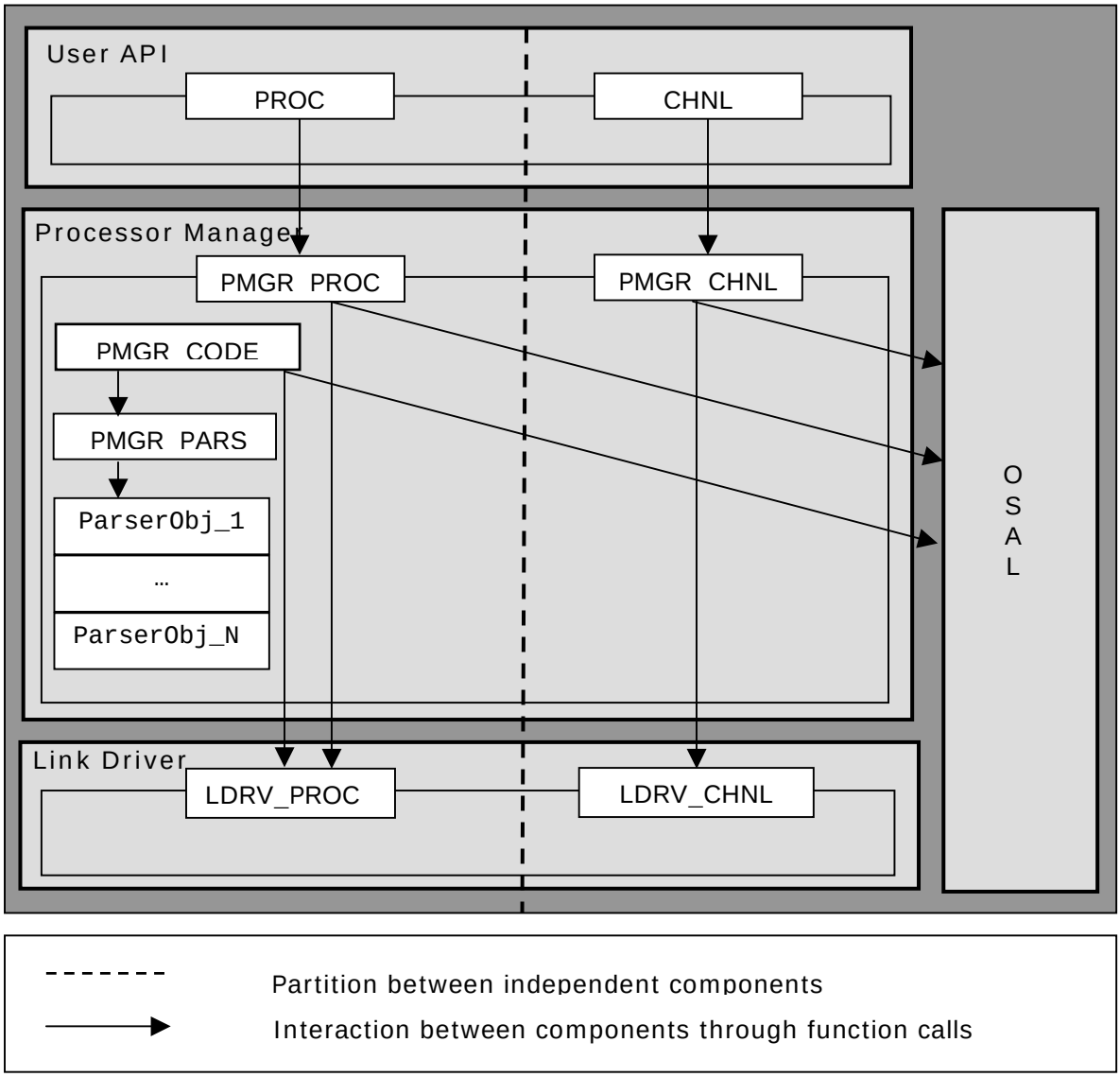

Figure 1. Relationship Between the Components in Processor Manager and DSP/BIOS™ Link

The PMGR_PROC subcomponent provides services to control the target DSP and uses services from PMGR_CODE and LDRV_PROC sub-components to accomplish its tasks.

The PMGR_CHNL component provides services for transferring data between the GPP and the DSP and uses the services that the LDRV_CHNL sub-component provides to accomplish its tasks.

The base image of a DSP is stored in COFF file format. PMGR_CODE uses the services that PMGR_PARS provides to parse the image and then loads this file onto the DSP. The PMGR_PARS sub-component is designed to be capable of understanding multiple COFF formats to support multiple and heterogeneous DSPs through DSP/BIOS™ Link. For this, it uses multiple (possibly plug-able) parsers.

3 PMGR_PROC

3.1 Resources Available

This subcomponent uses the services from the PMGR_CODE sub-component for parsing base image file and from LDRV_PROC for interacting/controlling the target DSP. It also uses OSAL for performing the OS dependent tasks in an OS independent manner.

3.2 Dependencies

3.2.1 Subordinates

PMGR_CODE, LDRV_PROC

3.2.2 Preconditions

PMGR_PROC_Attach() must be called before any other PMGR_PROC and PMGR_CHNL APIs are called.

3.3 Description

This subcomponent provides services to start, stop, and initialize a DSP. It also provides services to load a base image onto the target DSP. It maintains a list of clients that are attached to the DSP.

The first client (thread/process) that attaches to a DSP is designated as the owner of that DSP. Any number of clients can subsequently attach to and use the DSP. However, only the owner of the DSP has rights to load a base-image on the DSP and effect transitions in the DSP processor's state.

For example, from Idle to Loaded, Loaded to Started. (Refer to the Link Driver design document for details on the DSP's states).

PMGR_PROC releases the resources reserved for controlling the DSP only when the owner detaches from the DSP. Also, when the owner detaches from the DSP, all the other clients of the DSP are also detached and the DSP is in an unusable state i.e., is the 'Idle' state.

3.4 Typedefs and Data Structures

3.4.1 PMGR_ClientInfo

An element that holds process info and that can be manipulated using LIST.

Definition

```
typedef struct PMGR_ClientInfo_tag {
    ListElement   listElement ;
    PrcsObject  * prcsInfo    ;
} PMGR_ClientInfo ;
```

Fields

listElement	Structure that allows it to be used by LIST
prcsInfo	Placeholder for process information

Comments

None.

3.4.2 PMGR_PROC_SetupObj

Object containing information regarding setup of this subcomponent.

Definition

```
typedef struct PMGR_PROC_SetupObj_tag {
    Uint32         signature               ;
    PrcsObject   * owner                   ;
    SyncCsObject * mutex [MAX_PROCESSORS] ;
} PMGR_PROC_SetupObj ;
```

Fields

signature	Signature of this object
owner	Identifier of the owner of the subcomponent.
mutex	Critical section object to ensure mutual exclusion

Comments

None.

3.4.3 PMGR_PROC_Object

Object containing information maintained by this subcomponent.

Definition

```
typedef struct PMGR_PROC_Object_tag {
    Uint32       signature  ;
    PrcsObject * owner      ;
    List *       clients    ;
    Uint32       entryPoint ;
} PMGR_PROC_Object ;
```

Fields

`signature`	Signature of this object
`owner`	The owner of the processor
`clients`	List of clients that have attached to the processor
`entryPoint`	Entry point of the executable loaded on target processor

Comments

None.

3.5 API Definition

3.5.1 PMGR_PROC_Attach

Attaches the client to the specified DSP and also initializes the DSP (if required). The first caller to this function is designated as the owner of the DSP.

Syntax

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

Arguments

IN	ProcessorId	procId

Specifies the index of processor to attach to

OPT	ProcAttr *	attr

Attributes for the processor on which the attach must be done

Return Values

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

Comments

This function calls LDRV_PROC_Initialize () to initialize the DSP if it is not already initialized. This function maintains a list of client's process/thread IDs (as returned by PRCS_GetInfo ()) to keep track of all the clients attached to a target DSP.

Constraints

Build options can be specified to exclude PMGR_CHNL from the system. Therefore, this function initializes the PMGR_CHNL component conditionally.

See Also

PMGR_PROC_Detach

3.5.2 PMGR_PROC_Detach

This function allows the client to detach from a DSP and indicates the Processor Manager that the target DSP will not be used any longer.

Syntax

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

Arguments

IN	ProcessorId	procId

Identifier for the target DSP to be detached from.

Return Values

DSP_SOK	Operation completed successfully.
DSP_EFAIL	A failure occurred, unable to detach
DSP_ENOTOWNER	Not the owner of DSP
DSP_EATTACHED	Not attached to the target processor
DSP_EWRONGSTATE	Incorrect state to the completed requested operation

Comments

This function removes the caller's process/thread ID information from its list. If the caller is the owner of the target DSP, it releases all resources used for managing the DSP calls LDRV_PROC_Finalize().

Constraints

The callers must do a PMGR_PROC_Attach() before calling this function.

See Also

PMGR_PROC_Attach

3.5.3 PMGR_PROC_GetState

This function obtains the current state of the target DSP.

Syntax

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

Arguments

IN	ProcessorId	procId

DSP identifier.

OUT	ProcState *	ProcState

Buffer to hold the processor's current state. Link Driver defines this type.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid status buffer

Comments

This function queries the Link Driver to get the current state of DSP by querying the Link Driver. Since this function does not affect a state change on the DSP, all the clients are allowed to make a call to this function.

Constraints

The caller must do a PMGR_PROC_Attach() before calling this function.

See Also

PMGR_PROC_Load
PMGR_PROC_Start
PMGR_PROC_Stop
PMGR_PROC_Idle

3.5.4 PMGR_PROC_Load

This function loads the specified base image onto the target DSP.

Syntax

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

Arguments

IN ProcessorId procId

Target DSP identifier where the base image must load.

IN Char8 * imagePath

Full path to the image file to load on DSP

IN Uint32 argc

Number of argument to pass to the base image upon start

IN Char8 ** argv

Arguments to pass to the DSP main application

Return Values

DSP_SOK	Base image successfully loaded.
DSP_EACCESSDENIED	Not allowed to access the DSP
DSP_EFILE	Invalid base image
DSP_EFAIL	Unable to load image on DSP

Comments

Loads the specified base image onto the target DSP after ensuring that the caller is the owner of the target DSP. It invokes the services from the PMGR_CODE component for parsing the DSP image file, which loads the base image onto the DSP using the

LDRV_PROC interface. It also retrieves the start address of the base image and stores it in a private structure for future use (to be used in PMGR_PROC_Start()).

Constraints

The caller must do a PMGR_PROC_Attach() before calling this function.

See Also

PMGR_PROC_Attach
PMGR_PROC_LoadSection

3.5.5 PMGR_PROC_LoadSection

This function loads a particular section from the base image file onto the target DSP

Syntax

```
DSP_STATUS PMGR_PROC_LoadSection (ProcessorId  procId,
                                  FileName     imagePath,
                                  Uint32       sectID) ;
```

Arguments

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

Return Values

DSP_SOK	Operation successfully completed
DSP_EFILE	Invalid baseImage parameter
DSP_EINVALIDSECTION	Invalid section name
DSP_EACCESSDENIED	Not allowed to access the DSP
DSP_EFAIL	General failure, unable to load section on DSP

Comments

This function retrieves the specified section from the base image and loads it onto the target DSP using the services from PMGR_CODE

Constraints

The caller must do a PMGR_PROC_Attach() before calling this function.

See Also

PMGR_PROC_Attach
PMGR_PROC_Load

3.5.6 PMGR_PROC_Start

This function starts the execution of the loaded code on the DSP from the starting point specified in the base image.

Syntax

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

Arguments

IN	ProcessorId	procId

DSP identifier.

Return Values

DSP_SOK	Operation successfully completed
DSP_SALREATESTARTED	DSP is already in running state
DSP_EACCESSDENIED	Not allowed to access the DSP
DSP_EFAIL	General failure, unable to start the DSP
DSP_EATTACHED	Client has not attached the to the DSP

Comments

This function executes the loaded code on the DSP from the starting point specified in the base image. The function retrieves the start address of the base image when parsing the file (during PMGR_PROC_Load()).

Constraints

A base image must be loaded onto the target DSP before this call.

The caller must do a PMGR_PROC_Attach() before calling this function.

See Also

PMGR_PROC_Attach
PMGR_PROC_Load
PMGR_PROC_Stop

3.5.7 PMGR_PROC_Stop

The function stops the execution on the target DSP processor by making a call to LDRV_PROC_Stop ().

Syntax

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

Arguments

IN	ProcessorId	procId

DSP identifier.

Return Values

DSP_SOK	Operation successfully completed
DSP_SALREADYSTOPPED	DSP has stopped
DSP_EACCESSDENIED	Not allowed to access the DSP
DSP_EFAIL	General failure, unable to stop the DSP
DSP_EATTACHED	Client has not attached the to the DSP

Comments

None.

Constraints

The caller must do a PMGR_PROC_Attach() before calling this function.

See Also

PMGR_PROC_Attach
PMGR_PROC_Load
PMGR_PROC_Start

3.5.8 PMGR_PROC_Control

Provides a hook to perform device dependent control operations.

Syntax

```
DSP_STATUS PMGR_PROC_Control (ProcessorId dspId,
                              Int32       cmd,
                              Pvoid       arg) ;
```

Arguments

IN	ProcessorId	dspId
	Identifier for the DSP	
IN	Int32	cmd
	Command identifier.	
IN	Pvoid	arg
	Optional argument	

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid dspId or dspObj specified

Comments

None.

Constraints

`PMGR_Initialize ()` must be called before calling this function.

The DSP must not be in the `Error` state.

See Also

`None.`

3.5.9 PMGR_PROC_Debug

This function prints the current status of this component for debugging purposes

Syntax

```
Void PMGR_PROC_Debug () ;
```

Arguments

None.

Return Value

None.

Comments

None.

Constraints

None.

See Also

`PMGR_PROC_Attach`

3.5.10 PMGR_PROC_Instrument

Gets the instrumentation data associated with PMGR_PROC sub-component.

Syntax

```
DSP_STATUS PMGR_PROC_Instrument(ProcessorId      procId,
                                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 completed successfully
`DSP_EINVALIDARG`	`retVal` is invalid.

Comments

None.

Constraints

procId must be valid.

retVal must be a valid pointer.

See Also

None.

3.5.11 PMGR_PROC_IsAttached

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

Syntax

```
PMGR_PROC_IsAttached (ProcessorId  procId,
                      PrcsObject * client,
                      Bool * isAttached) ;
```

Arguments

IN	ProcessorId	procId

Identifier for processor for which instrumentation information is to be obtained.

OUT	PrcsObject *	client

Client identifier.

OUT	Bool *	isAttached

Placeholder for flag indicating the client is attached.

Return Values

DSP_SOK	Operation completed successfully
DSP_EINVALIDARG	Invalid argument

Comments

None.

Constraints

procId must be valid.

See Also

PMGR_PROC_Attach

3.5.12 PMGR_PROC_Destroy

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

Syntax

```
Void PMGR_PROC_Destroy () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation completed successfully
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	DSP_EFAIL

Comments

None.

Constraints

None.

See Also

PMGR_PROC_Setup

3.5.13 PMGR_PROC_Setup

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

Syntax

```
Void PMGR_PROC_Destroy () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation completed successfully
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

Comments

None.

Constraints

None.

See Also

PMGR_PROC_Destroy

4 PMGR_CHNL

4.1 Resources Available

This component uses the services from the LDRV_CHNL and OSAL components to achieve its tasks.

4.1.1 Subordinates

None.

4.1.2 Preconditions

PMGR_PROC_Attach () must be done before making any calls from this component

4.2 Description

This component provides the infrastructure to transfer the data buffers between the DSP and the GPP. The current design restricts the usage of a channel by only one process/thread.

4.3 API Definition

4.3.1 PMGR_CHNL_Initialize

Sets up all channel objects in Link Driver.

Syntax

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

Arguments

IN	ProcessorId	procId
	Processor ID	

Return Values

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

Comments

This function calls LDRV_CHNL_Initialize () to set up all the channel objects in the Link Driver.

Constraints

ProcessorId must be valid.

See Also

PMGR_CHNL_Finalize
PMGR_CHNL_Create

4.3.2 PMGR_CHNL_Finalize

Releases all channel objects setup in Link Driver.

Syntax

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

Arguments

IN	ProcessorId	procId
	Processor ID	

Return Values

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

Comments

None.

Constraints

Channels for specified processor must be initialized. Processor Id must be valid.

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_Destroy

4.3.3 PMGR_CHNL_Create

Creates resources used for transferring data between GPP and DSP.

Syntax

```
DSP_STATUS PMGR_CHNL_Create (ProcessorId  procId,
                             ChannelId    chnlId,
                             ChnlAttrs *  attrs);
```

Arguments

IN	ProcessorId	procId
	Processor ID	
IN	ChannelId	chnlId
	Channel ID of channel to create	
IN	ChnlAttrs *	attrs
	Channel attributes, if NULL, default attributes are applied	

Return Values

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

Comments

This function calls LDRV_CHNL_Open ( ) and creates the resources for transferring the data between the GPP and the DSP.

Constraints

Channels for specified processors must be initialized. Processor and channel ids must be valid. Attributes must be valid.

See Also

PMGR_CHNL_Initialize

4.3.4 PMGR_CHNL_Delete

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

Syntax

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

Arguments

IN	ProcessorId	procId
	Processor Identifier	
IN	ChannelId	chnlId
	Channel Identifier	

Return Values

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

Comments

None.

Constraints

Channels for specified processors must be initialized. Processor and channel ids must be valid.

See Also

PMGR_CHNL_Create

4.3.5 PMGR_CHNL_AllocateBuffer

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

Syntax

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

Arguments

IN	ProcessorId	procId
	Processor Identifier	
IN	ChannelId	chnlId
	Channel Identifier	
OUT	Char8 **	bufArray
	Pointer to receive an array of allocated buffers	

IN	Uint32	size
	Size of each buffer	
IN	Uint32	numBufs
	Number of buffers to allocate	

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General failure, channel not initialized
DSP_EMEMORY	Operation failed due to memory error

Comments

None.

Constraints

Channels for specified processors must be initialized. Processor and channel ids must be valid.

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_FreeBuffer

4.3.6 PMGR_CHNL_FreeBuffer

Frees buffer(s) allocated by PMGR_CHNL_AllocateBuffer.

Syntax

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

Arguments

IN	ProcessorId	procId
	Processor ID	
IN	ChannelId	chnlId
	Channel ID	
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 completed successfully
DSP_EFAIL	General failure, channel not initialized
DSP_EMEMORY	Operation failed due to memory error

Comments

None.

Constraints

Channels for specified processors must be initialized. Processor and channel ids must be valid.

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_AllocateBuffer

4.3.7 PMGR_CHNL_Issue

Issues an input or output request on a specified channel.

Syntax

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

Arguments

IN	ProcessorId	procId
	Processor Identifier	
IN	ChannelId	chnlId
	Channel Identifier	
IN	ChannelIOInfo *	ioReq
	IO request packet	

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General failure
DSP_EMEMORY	Operation failed due to memory error
DSP_EACCESSDENIED	Not the owner of the channel

Comments

This function calls LDRV_CHNL_AddIORequest() to queue ioReq on the channel.

Constraints

Channels for specified processors must be initialized. Processor and channel ids must be valid.

See Also

PMGR_CHNL_Reclaim

4.3.8 PMGR_CHNL_Reclaim

Gets the buffer back that has been issued to this channel

Syntax

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

Arguments

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

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General failure, channel not initialized
DSP_EMEMORY	Operation failed due to memory error
DSP_EACCESSDENIED	Not the owner of the channel
DSP_ETIMEOUT	Timed out. Waiting for a buffer on channel
CHNL_E_NOIOC	Timeout parameter was "NO_WAIT", yet no I/O completions were queued.

Comments

This function calls LDRV_CHNL_AddIORequest() to queue ioReq on the channel.

Constraints

Channels for specified processors must be initialized. Processor and channel ids must be valid.

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_AllocateBuffer

4.3.9 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

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

Arguments

IN	ProcessorId	procId
	Processor ID	
IN	ChannelId	chnlId
	Channel ID	

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General failure, channel not initialized
DSP_EMEMORY	Operation failed due to memory error
DSP_EACCESSDENIED	Not the owner of the channel
DSP_ETIMEOUT	Time out occurred before the channel could be idled

Comments

None.

Constraints

Channels for specified processor must be initialized. Processor and channel ids must be valid.

See Also

PMGR_CHNL_Initialize
PMGR_CHNL_Create

4.3.10 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

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

Arguments

IN	ProcessorId	procId
	Processor Identifier	
IN	ChannelId	chnlId
	Channel Identifier	

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General failure, channel not initialized
DSP_EMEMORY	Operation failed due to memory error

Comments

None.

Constraints

Channels for specified processor must be initialized. Processor and channel ids must be valid.

See Also

```
PMGR_CHNL_Initialize
PMGR_CHNL_Create
PMGR_CHNL_Issue
```

4.3.11 PMGR_CHNL_Control

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

Syntax

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

Arguments

IN	ProcessorId	procId
	Processor Identifier	
IN	ChannelId	chnlId
	Channel Identifier	
IN	Int32	cmd
	Command id.	

IN	Pvoid	arg
	Optional argument	

Return Values

DSP_SOK	Operation completed successfully
DSP_ENOTIMPL	Functionality not implemented

Comments

This function provides a hook to perform the device dependent control operations on channels. Not implemented in current implementation

Constraints

None.

See Also

PMGR_CHNL_Initialize

4.3.12 PMGR_CHNL_Debug

This function prints the current status of the PMGR_CHNL sub-component.

Syntax

```
Void PMGR_CHNL_Debug () ;
```

Arguments

None.

Return Value

None.

Comments

None.

Constraints

None.

See Also

None.

4.3.13 PMGR_CHNL_Instrument

Gets the instrumentation information related to CHNL's

Syntax

```
PMGR_CHNL_Instrument (ProcessorId       procId,
                      ChannelId         chnlId,
                      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 completed successfully.
DSP_EINVALIDARG	retVal is invalid.

Comments

This function provides a hook to perform the device dependent control operations on channels. Not implemented in current implementation.

Constraints

retVal must be a valid pointer

See Also

None.

5 PMGR_CODE

5.1 Description

This component provides the COFF file parsing services to the DSP/BIOS™ Link.Link is designed to support heterogeneous DSPs and therefore this component creates different parser objects to handle this scenario.

Based on the CFG information of Link, PMGR_CODE modifies itself and can load parsers for different file formats. A call to PMGR_CODE_LoadExecutable() results in multiple calls to the PMGR_PARS sub-component functions. These functions in turn load the data into format independent structures that are used while loading the image onto the DSP.

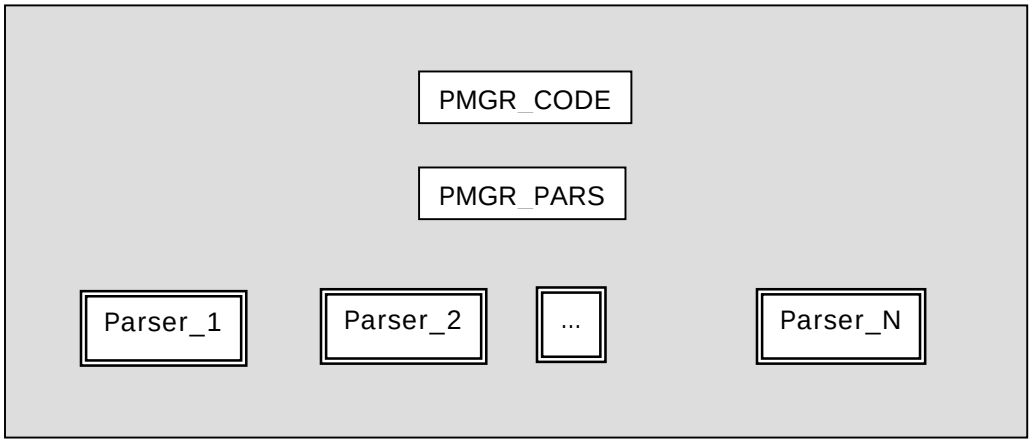

Figure 2. Components involved in parsing a DSP executable.

5.2 API Definition

5.2.1 PMGR_CODE_LoadExecutable

Uses interfaces provided in ParserObj to parse the COFF file and load it onto DSP.

Syntax

```
DSP_STATUS PMGR_CODE_LoadExecutable (ProcessorId  procId,
                                     FileName     baseImage,
                                     Uint32       argc,
                                     Char8 **     argv,
                                     Uint32 *     entryAddress) ;
```

Arguments

IN	ProcessorId	procId

Target DSP identifier where the base image is to load

IN	FileName	baseImage

File identifier for the base image

IN	Uint32	argc

Number of arguments to pass to the base image upon start

IN	Char8 **	argv

Arguments to pass to the DSP main application.

OUT	Uint32 *	entryAddress

OUT argument for returning entry address for the executable

Return Values

DSP_SOK	Base image successfully loaded
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 procId argument.

Comments

None.

Constraints

procId must be a valid DSP processor ID.

baseImage must be a valid file identifier.

entryAddress must be a valid section identifier.

See Also

PMGR_PROC_Load

5.2.2 PMGR_CODE_LoadSection

Uses interfaces provided in ParserObj to parse the COFF file and load it onto DSP.

Syntax

```
DSP_STATUS PMGR_CODE_LoadSection (ProcessorId  procId,
                                  FileId *     baseImage,
                                  Uint32       sectId) ;
```

Arguments

IN	ProcessorId	procId
	DSP identifier	
IN	FileId *	baseImage
	Full path to the image file.	
IN	Uint32	sectId
	Identifier for the 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 procId argument.
DSP_EINVALIDSECT	Invalid section name

Comments

None.

Constraints

procId must be a valid DSP processor ID.

baseImage must be a valid file identifier.

sectId must be a valid section identifier.

See Also

PMGR_PROC_Load

5.2.3 PMGR_CODE_Debug

This function prints the current status of the PMGR_CODE sub-component.

Syntax

```
Void PMGR_CODE_Debug () ;
```

Arguments

None.

Return Value

None.

Comments

None.

Constraints

None.

See Also

None.

6 PMGR_PARS

6.1 Resources Available

This subcomponent uses services from the parser to get image data in format dependent structures.

6.1.1 Subordinates

None.

6.1.2 Preconditions

None.

6.2 Description

This subcomponent provides the PMGR_CODE subcomponent with image data in format independent structures to use while loading the image onto the DSP.

6.3 Typedefs and Data Structures

6.3.1 ImageAttributes

This structure defines a format agnostic definition of attributes that a parser requires.

Definition

```
typedef struct ImageArttributes_tag {
    Uint16   version           ;
    Uint16   numSections       ;
    Int32    symTabOffset      ;
    Int32    numSymTabEntries  ;
    Uint16   numBytesOptHeader ;
    Uint16   flags             ;
    Uint16   targetId          ;
} ImageAttributes ;
```

Fields

version	The version of the file format
numSections	Number of sections in a file
symTabOffset	Symbol table offset in a file
numSymTabEntries	Number of symbol table entries in a file
numBytesOptHeader	Number of bytes in the optional header
flags	Flags associated with the file format
targetId	Target of the DSP base image file

6.3.2 OptImageAttributes

Structure defining a format agnostic definition of optional attributes required from a parser. This structure is a placeholder for optional attributes associated with file. These attributes could be useful in debugging.

Definition

```
typedef struct OptImageAttributes _tag {
      Int32       dummy ;
} OptImageAttributes ;
```

Fields

dummy	Dummy parameter (unused)

6.3.3 SectionAtrributes

Structure defining a format agnostic definition of section related attributes required from a parser.

Definition

```
typedef struct SectionAttributes_tag {
    Char8 * name          ;
```

```
    Uint32  index         ;
    Uint32  size          ;
    Uint32  sectOffset    ;
    Uint32  loadAddr      ;
    Uint32  runAddr       ;
    Bool    isLoadSection ;
    Char8 * data          ;
} SectionAttributes ;
```

Fields

name	Name of the section
index	Index of the section in the DSP base image file
size	Size of the section data in bytes
sectOffset	Offset of the section data in a file
loadAddr	Load address of the section data
runAddr	Run address of the section
isLoadSection	Flag to indicate that the section is loadable
data	Buffer to hold data

6.3.4 SymbolAttrs

This structure defines the format agnostic definition of symbols and their attributes.

Definition

```
typedef struct SymbolAttrs_tag {
    Uint32  symIndex ;
    Char8 * name     ;
    Uint32  addr     ;
} SymbolAttrs ;
```

Fields

symIndex	Index of the symbol in the symbol table
name	Name of the symbol
addr	Address of the symbol

6.3.5 ParserContext

This structure defines the context of parser. 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.

Definition

```
typedef struct ParserContext_tag {
    KFileObject *        fileObj    ;
    ProcessorId          procId     ;
```

```
    Uint32               startAddr  ;
    ImageAttributes *    attrs      ;
    OptImageAttributes * optAttrs   ;
    Uint32               numSymbols ;
    SymbolAttrs *        symbols    ;
} ParserContext ;
```

Fields

`fileObj`	File object for the DSP base image file
`procId`	Processor identifier
`startAddr`	Entry point address for the DSP base image file
`attrs`	Attributes associated with the DSP base image file
`optAttrs`	Optional attributes associated with the DSP base image file
`numSymbols`	Number of symbols in the DSP base image file
`symbols`	Symbol table containing all the symbols from the DSP base image file

6.4 API Definition

6.4.1 PMGR_PARS_Initialize

Initializes a base image file for parsing. This function is required to be called before any other function is called from this sub-component.

Syntax

```
DSP_STATUS PMGR_PARS_Initialize (ProcessorId  procId,
                                 FileName     file,
                                 Void **      obj) ;
```

Arguments

IN	ProcessorId	procId

Processor Id

IN	FileName	file

Identifier for the file.

OUT	Void **	obj

OUT argument that contains the object to be passed in any subsequent call from this subcomponent.

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Memory error

Comments

None.

Constraints

file must be valid.

See Also

PMGR_PARS_Finalize

6.4.2 PMGR_PARS_Finalize

This function releases the context object obtained through PMGR_PARS_Initialize.

Syntax

```
DSP_STATUS PMGR_PARS_Finalize (Pvoid  objCtx) ;
```

Arguments

IN	Pvoid	objCtx

The context object that PMGR_PARS_Initialize() obtains

Return Values

DSP_SOK	Operation completed successfully
DSP_EMEMORY	Operation failed due to memory error

Comments

None.

Constraints

objCtx must be valid.

See Also

PMGR_PARS_Initialize

6.4.3 PMGR_PARS_GetImageAttributes

This function gets the attributes for a particular base image file.

Syntax

```
DSP_STATUS PMGR_PARS_GetImageAttributes (Pvoid            objCtx,
                                         ImageAttributes ** attrs);
```

Arguments

IN	Pvoid	objCtx	
	The context object that PMGR_PARS_Initialize () obtains		
OUT	ImageAttributes **	attrs	
	Required attributes associated with the DSP base image file		

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File format not supported
DSP_ERANGE	File seek operation failed
DSP_EMEMORY	Operation failed due to memory error
DSP_EINVALIDARG	Invalid arguments

Comments

None.

Constraints

objCtx must be valid..

See Also

PMGR_PARS_Initialize

6.4.4 PMGR_PARS_GetOptImageAttributes

This function gets the optional attributes for a particular base image file.

Syntax

```
DSP_STATUS
PMGR_PARS_GetOptImageAttributes (Pvoid                 objCtx,
                                 OptImageAttributes ** optattrs) ;
```

Arguments

IN	Pvoid	objCtx	
	The context object that PMGR_PARS_Initialize () obtains		
OUT	OptImageAttributes **	optattrs	
	Optional attributes associated with the DSP base image file		

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File format not supported
DSP_ERANGE	File seek operation failed
DSP_EMEMORY	Operation failed due to memory error
DSP_EINVALIDARG	Invalid arguments

Comments

None.

Constraints

objCtx must be valid.

See Also

PMGR_PARS_Initialize

6.4.5 PMGR_PARS_GetEntryAddress

Gets the entry address for a particular base image file

Syntax

```
DSP_STATUS PMGR_PARS_GetEntryAddress (Pvoid    objCtx,
                                      Uint32 * addr) ;
```

Arguments

IN	Pvoid	objCtx	
	The context object obtained through PMGR_PARS_Initialize ()		
OUT	Uint32 *	addr	
	OUT argument containing the entry address for the base address		

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File format not supported
DSP_ERANGE	File seek operation failed
DSP_EMEMORY	Operation failed due to memory error
DSP_EINVALIDARG	Invalid arguments

Comments

None.

Constraints

objCtx must be valid.

See Also

PMGR_PARS_Initialize

6.4.6 PMGR_PARS_GetSymbolAddress

This function gets the address of a particular symbol.

Syntax

```
DSP_STATUS PMGR_PARS_GetEntryAddress (Pvoid    objCtx,
                                      Char8 *  symName,
                                      Uint32 * addr) ;
```

Arguments

IN	Pvoid	objCtx	
	The context object that PMGR_PARS_Initialize () obtains		
IN	Char8 *	symName	
	Name of the symbol		
OUT	Uint32 *	addr	
	OUT argument containing the entry address for the base address		

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File format not supported
DSP_ERANGE	File seek operation failed
DSP_EMEMORY	Operation failed due to memory error
DSP_EINVALIDARG	Invalid arguments

Comments

None.

Constraints

objCtx must be valid.

symName must be valid.

See Also

PMGR_PARS_Initialize

6.4.7 PMGR_PARS_GetSectionAttributes

Gets the attributes associated with a section. Memory for holding the section attributes must be allocated by the caller.

Syntax

```
DSP_STATUS PMGR_PARS_GetEntryAddress(Pvoid          objCtx,
                                     Uint32         sectIndex,
                                     SectionAttributes* sectAttrs);
```

Arguments

IN	Pvoid	objCtx

The context object that PMGR_PARS_Initialize () obtains

IN	Uint32	sectIndex

Index of the section

OUT	SectionAttributes *	sectAttrs

OUT argument containing the attributes associated with a section

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File format not supported
DSP_ERANGE	File seek operation failed
DSP_EMEMORY	Operation failed due to memory error
DSP_EINVALIDARG	Invalid arguments

Comments

None.

Constraints

objCtx must be valid pointer.

sectAttrs must be a valid pointer.

The data field in sectAttrs must be a valid buffer.

See Also

PMGR_PARS_Initialize
PMGR_PARS_GetSectionAttributes

6.4.8 PMGR_PARS_GetSectionData

This function gets the data for a section.

Syntax

```
DSP_STATUS PMGR_PARS_GetSectionData (Pvoid    objCtx,
                                     SectionAttributes * sectAttrs) ;
```

Arguments

IN	Pvoid	objCtx

The context object through PMGR_PARS_Initialize

IN OUT	SectionAttributes *	sectAttrs

IN OUT argument containing the section attributes with section data

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File format not supported
DSP_ERANGE	File seek operation failed
DSP_EMEMORY	Operation failed due to memory error
DSP_EINVALIDARG	Invalid arguments

Comments

None.

Constraints

objCtx must be valid pointer.

sectAttrs must be a valid pointer.

The data field in sectAttrs must be a valid buffer.

See Also

PMGR_PARS_Initialize
PMGR_PARS_GetSectionAttributes

6.4.9 PMGR_PARS_Debug

This function prints the current status of the PMGR_PARS component.

Syntax

```
Void PMGR_PARS_Debug () ;
```

Arguments

None.

Return Value

None.

Comments

None.

Constraints

None.

See Also

None.

6.4.10 PMGR_PARS_FillArgsBuffer

Fills up the data-buffer with the specified arguments to be sent to DSP's "main" function.

Syntax

```
PMGR_PARS_FillArgsBuffer (ProcessorId          procId,
                          Uint32               argc,
                          Char8 **             argv,
SectionAttributes *  sectAttrs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier	
IN OUT	SectionAttributes *	sectAttrs
	Attributes of the ".args" section	
IN	Uint32	argc
	Number of arguments to be passed	
IN	Char8 **	argv
	Argument strings to be passed.	

Return Values

DSP_SOK	Operation completed successfully
DSP_ESIZE	Insufficient space in .args buffer to hold all the arguments
DSP_EMEMORY	Operation failed due to memory error.

Comments

None.

Constraints

`ProcessorId` must be valid.

`argc` must be more than 0.

`argv` must be valid pointer.

`sectAttrs` must be a valid pointer.

See Also

None.

7 Appendix

7.1 Concept of Ownership of Components

The concept of ownership in DSP/BIOS™ LINK is defined as:

1. The first user of an instance of a component is designated as the owner for that instance.
2. All the resources used for managing/interfacing the component are released when the owner releases the component.
3. If the owner releases the component, the associated resources are released even when other clients have not released the component.

This is different compared to the 'lock' interface implementation. The 'lock' mechanism allows a client to specify the access rights that it wants.

The current design allows a much simpler way to control the ownership of a component. Especially for `PMGR_PROC`, as the first client is designated as the owner, it simplifies the user side implementation. The client that gets a return code of `DSP_SALREADYATTACHED` can safely assume that some other client has already attached to the DSP and loaded the base image. Also, since state transitions can occur from only one place, the user side code is simplified.

7.2 Future Enhancements

DSP/BIOS™ LINK currently allows a channel to be accessed from only one thread. As a future enhancement, the plan is to allow multiple threads to share a channel for data communication. Threads that belong to a process context can be assumed to be coordinating threads and can be allowed to share a channel. However, we can have a restriction that two processes cannot access the same channel.

In this scenario as well, the first thread that opens a channel can be designated as the owner of that channel. Other threads can also open the same channel but when the owner closes the channel (by a call to `PMGR_CHNL_Close()`) it is unusable.