DSP/BIOS™ LINK

MPLIST DESIGN

LNK 131 DES

Version 0.50

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

TABLE OF FIGURES

1 Introduction

1.1 Purpose & Scope

This document describes the design and interface definition of linked list based transport mechanism between GPP and DSP.

The document is targeted at the development team of DSP/BIOS™ LINK.

1.2 Terms & Abbreviations

DSPLINK	DSP/BIOS™ LINK
MPLIST	Multi-processor list
SMA	Shared Memory Allocator
☞	This bullet indicates important information. Please read such text carefully.
❑	This bullet indicates additional information.

1.3 References

1.	LNK 084 PRD	DSP/BIOS™ LINK Product Requirement Document
2.	LNK 082 DES	POOL Design Document
3.	LNK 132 DES	PCI Driver Redesign

1.4 Overview

DSP/BIOS™ LINK is runtime software, analysis tools, and an associated porting kit that simplifies the development of embedded applications in which a general-purpose microprocessor (GPP) controls and communicates with a TI DSP. DSP/BIOS™ LINK provides control and communication paths between GPP OS threads and DSP/BIOS™ tasks, along with analysis instrumentation and tools.

This module provides the design for implementing a linked-list in the pool.

This document describes the various design alternatives to achieve the linked-list functionality between GPP and DSP using DSP/BIOS™ LINK. It also gives an overview of the MPLIST component on the GPP and DSP-sides of *DSPLINK* and its interaction with the other components within *DSPLINK*. The document also gives a detailed design with sequence diagrams of MPLIST component.

2 Requirements

Please refer to section 16.2 of LNK 084 PRD - DSP/BIOS™ LINK Product Requirement Document.

This module provides a linked list based transport mechanism between GPP and DSP.

On the devices where a shared memory region exists between GPP and DSP, this module shall implement the linked-list in the shared memory region. In cases where a shared memory region does not exist, the module shall internally maintain coherence between linked lists on the remote processors.

R105	The APIs shall enable users to create a linked list that can be used as a transport between GPP and DSP.
R106	This module shall support variable size buffers to be transferred between GPP and DSP. These buffers shall be required to contain a header structure of 8 bytes in addition to buffer contents.
R107	This module shall support placing an element at the end of list and removing the element from the front of the list.
R108	This module shall also enable applications to insert a buffer at any location in the linked-list. The location shall be identified through an existing node in the linked-list.
R109	This module shall also enable applications to remove a buffer from any location in the linked-list. This location shall be identified through the node to be removed.

In addition, the linked-list based transport mechanism must meet the following generic requirement:

1. The API exported by the linked-list component shall be common across different GPP operating systems.
2. Both the DSP as well as the GPP side shall expose same API.
3. During static configuration the priority shall be assigned to the linked-list based transport mechanism. It will have the lowest priority below other DSP/BIOS™ LINK components.
4. Multiple threads can perform linked-list operations on the linked-lists created in the system. However, ownership shall come into play while creation/deletion of the list. The processor which creates the linked-list shall be the one which deletes it.

3 Assumptions

The MPLIST design makes the following assumptions:

1. The hardware provides a buffer pool, to which both the GPP and the DSP have access.

4 Constraints

The user constraints are:

- The size of the buffer transported by the linked-list should be at least greater than the size of the fixed linked-list element header.
- Each linked-list will have an associated critical section as the data resides in pool and multiple processors can access it.
- The linked-list elements must have a fixed header as their first field. This header is used by the MPLIST component for including information required for accessing the element. The contents of the linked-list header are reserved for use internally within *DSPLINK* and should not directly be modified by the user.
- The linked-list elements must be allocated and freed through POOL APIs provided by *DSPLINK*. Elements allocated through the POOL API can be accessed by multiple processors. Any other means for memory allocation (for example: standard OS calls) will fail as the elements cannot be accessed across processors.
- The user has to use unique identifier to identify individual linked-lists across the system.
- The user can use the linked-list only as a transport mechanism between GPP and DSP and not among different processors as the linked-list works on underlying POOL concept.

5 High Level Design

In a multiprocessor system having shared access to a memory region, a linked list based transport mechanism between GPP and DSP can be implemented. . In cases where a shared memory region does not exist, the module shall internally maintain coherence between linked lists on the remote processors. This mode of communication is called MPLIST transfer mode.

MPLIST module will expose linked-list based transport mechanism to the user. It shall provide a mechanism for creating the list, common list operations like addition/removal of element and deletion of list. The list will be created using POOL. This will ensure both GPP/DSP have visibility into the list. The client will have to provide a list element structure contained fixed size header information along with variable size buffers which can be added to the list.

Memory needed for creation of the list shall be taken from the buffer pool. After freeing the list the memory shall be returned to the buffer pool. The linked-list component utilizes buffer pool which can be configured through the static configuration system.

The basic data structures of linked-list based transport mechanism from a client's perspective are a linked-list and linked-list element. Each linked-list shall be addressed through a unique name.

The MPLIST component on each processor shall provide the ability to create and delete a list from the buffer pool. It shall also provide the client with means of traversing, writing into and reading from the list.

Linked list related operations like

1. Creating a linked-list that can be used as a transport mechanism between GPP and DSP.
2. Placing an element at the tail of list.
3. Getting an element from the head of the list.
4. Removing an element from the list.
5. Traversing the list.
6. Insert an element before any location in the linked-list.
7. Deleting a linked-list.

5.1 Architecture overview

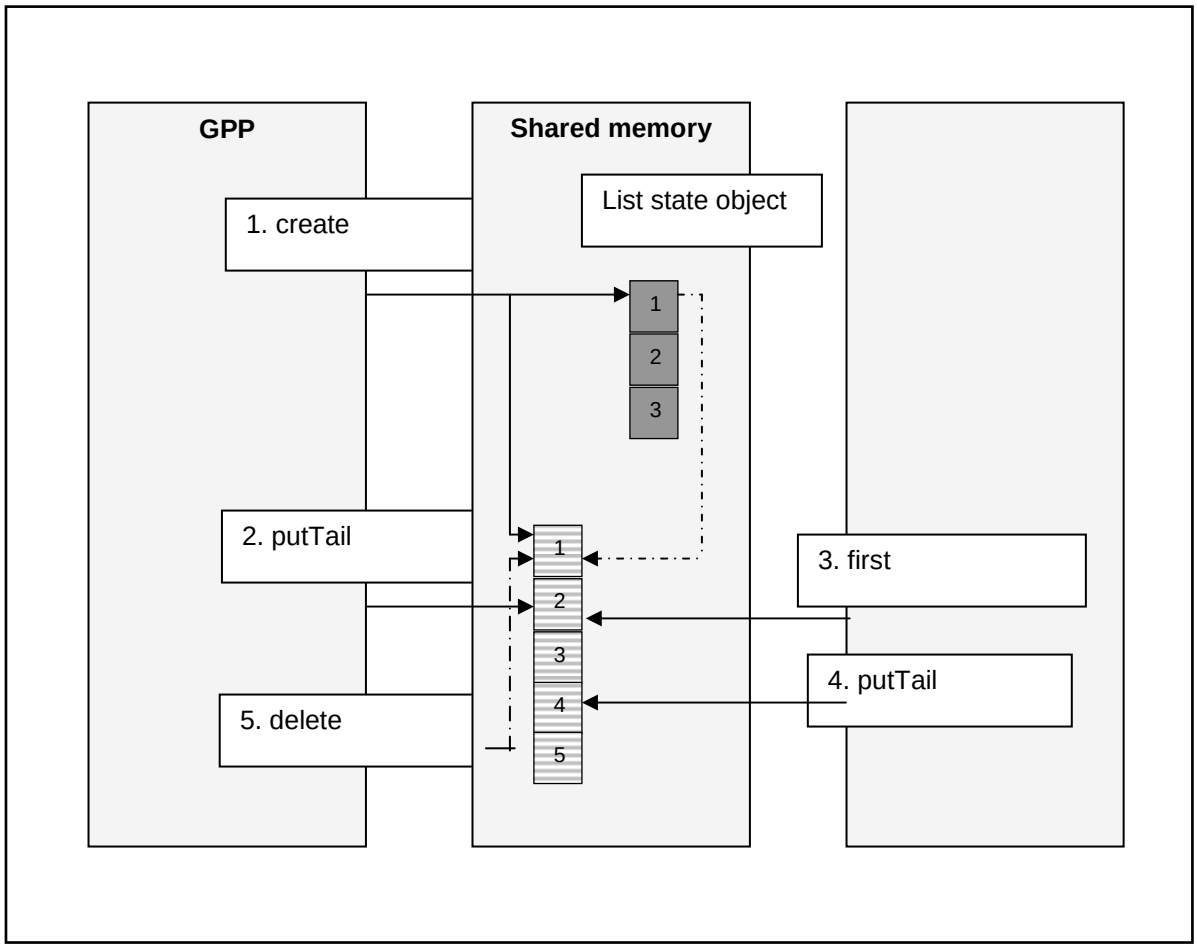

Figure 1. Basic architecture of system supporting linked-list transfer mode

Linked-List

The list that will be used in the linked-list based transport mechanism is a circular doubly linked list.

Linked-List element

The list can contain variable sized list elements.

The element must contain the fixed linked-list header as the first element. This header is not modified by the user, and is used within *DSPLINK* for including information required for the linked-list.

APIs provided by the linked-list component are used for allocating and freeing the linked-list element. Pool has to be specified for allocation of the linked-list element. Linked-list elements cannot be allocated on the stack or directly through the standard OS allocation and free functions.

Initialization and finalization

Before using any of the MPLIST features, the user must initialize the POOL component.

A global object which contains information about all linked lists is present in the system. This global object is termed as the list state object. This global object is used among multiple processors to see which lists are present in the system. This object also enables any processor to retrieve the linked-list handle to which it wants to perform linked-list operations. The system internally needs to initialize this global object.

Internal MPLIST functions like _MPLIST_init and _MPLIST_exit shall be used to initialize and finalize the system respectively.

Creating a linked-list

Both the GPP and the DSP can create the linked-list. The only constraint is that the processor which creates the linked-list must delete it.

The memory needed to create the linked-list is obtained from buffer pool. After translation of the address (if required) from the kernel space into the user space, the pointer to the list structure is returned to the user application that had requested the buffer allocation.

On creation of the list, the global list state object is updated with an entry to the newly created list. This entry is used by other processors when they want to perform linked-list operations on any list.

When a linked-list is created or referenced, a unique handle to the linked-list is returned to the user. This handle is used for all further accesses to the linked-list. The unique linked-list handle is a structure which contains the handle to the linked list, a critical section and the entry with the global list state object

On the DSP-side, the list creation proceeds in a similar manner.

Use of API MPLIST_create can be made to create the list.

Traversing a linked-list

The client uses the linked-list handle from the global list state object to reference the linked-list to traverse the list.

With the linked-list handle and use of API's like MPLIST_first and MPLIST_next link traversal can be achieved.

Deleting a linked-list

Both the GPP and the DSP can delete the linked-list. The constraints to be kept in mind while deleting the linked-list are:

1. The processor which creates the list shall delete it.

This implies that the DSP cannot delete a list created by the GPP and vice versa.

On a GPP-side call to free a list, the buffer pool takes care of the address translation needed between GPP user and kernel space internally. The list to be deleted must be empty as the individual list elements will not be deleted. On deletion of the list, the global list state object is updated by removing the entry to the deleted list.

On the DSP-side, the list is freed in a similar manner.

Use of API MPLIST_delete can be made to delete the list.

Adding an element to the list

The element can be added to the tail of the list. An API can also be provided where the client can insert an element before a particular list element.

The client uses the linked-list handle from the global list state object to reference the linked-list to which it wants to add an element.

Use of API's MPLIST_putTail and MPLIST_insertBefore can be made to insert an element to the list.

Removing an element from the list

The client uses the linked-list handle from the global list state object to reference the linked-list from which it wants to remove an element.

An API is provided where the list element to be removed itself is passed. Use of API MPLIST_removeElement can be made to remove an element from the list.

Existing components like POOL and MPCS will be used to implement linked-list. New components MPLIST (which exposes the API used by the client) will be additionally created.

This design assumes hardware provides a buffer pool, to which both the GPP and the DSP have access. In some systems, shared memory is the buffer pool to which both GPP and DSP have access. When shared memory is not present, other methods like a combination of signaling and DMA access can be used. Please refer to LNK_132_DES design document.

5.2 Control flow

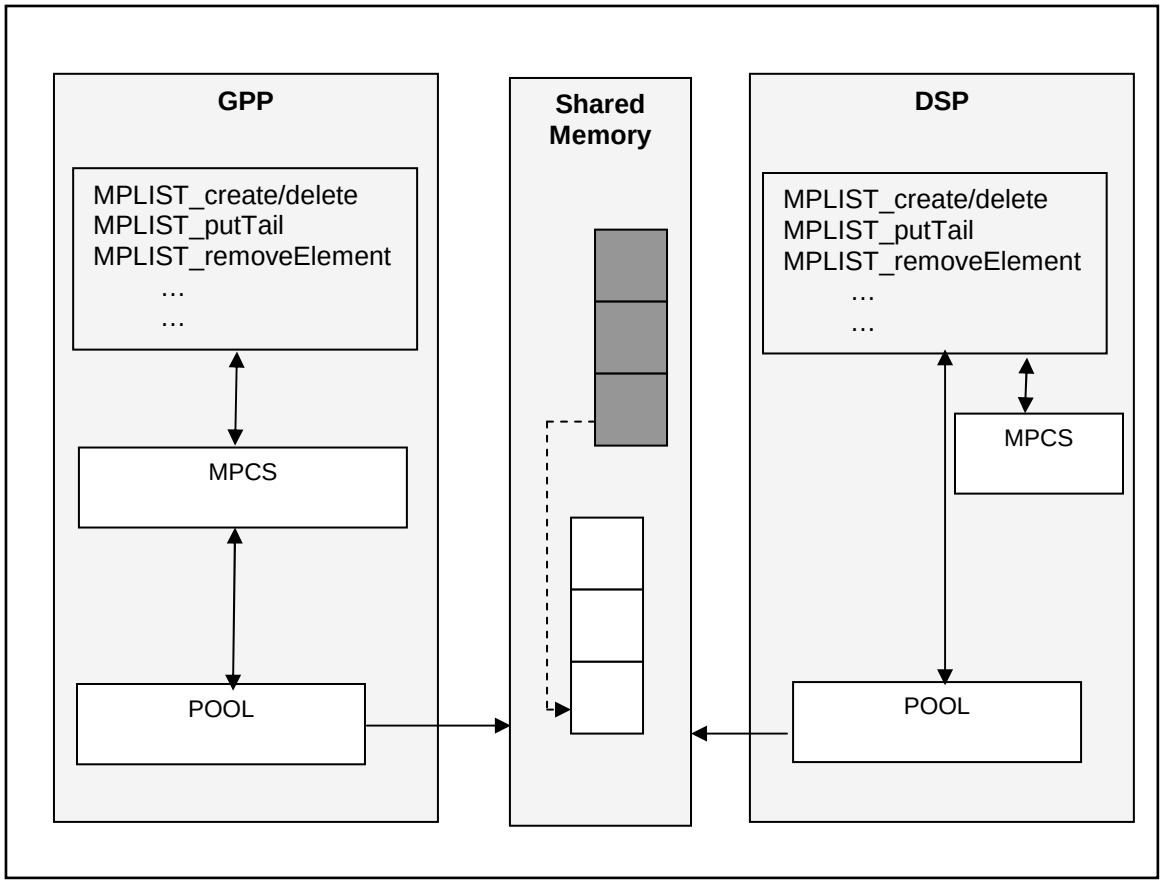

Figure 2. MPLIST sub-component interaction

6 Sequence Diagrams

The following sequence diagrams show the control flow for a few of the important functions to be implemented within the *DSPLINK* MPLIST driver.

The sequence diagrams indicate the linked-list control flow through the MPLIST component and its interaction with the rest of the *DSPLINK* components and the MPLIST components on the DSP-side.

Sequence diagrams have been drawn for the control flow on the GPP as well as the DSP side.

- *The dashed arrow in all sequence diagrams indicates an indirect control transfer, which does not happen through a direct function call.*

6.1 _MPLIST_init ()

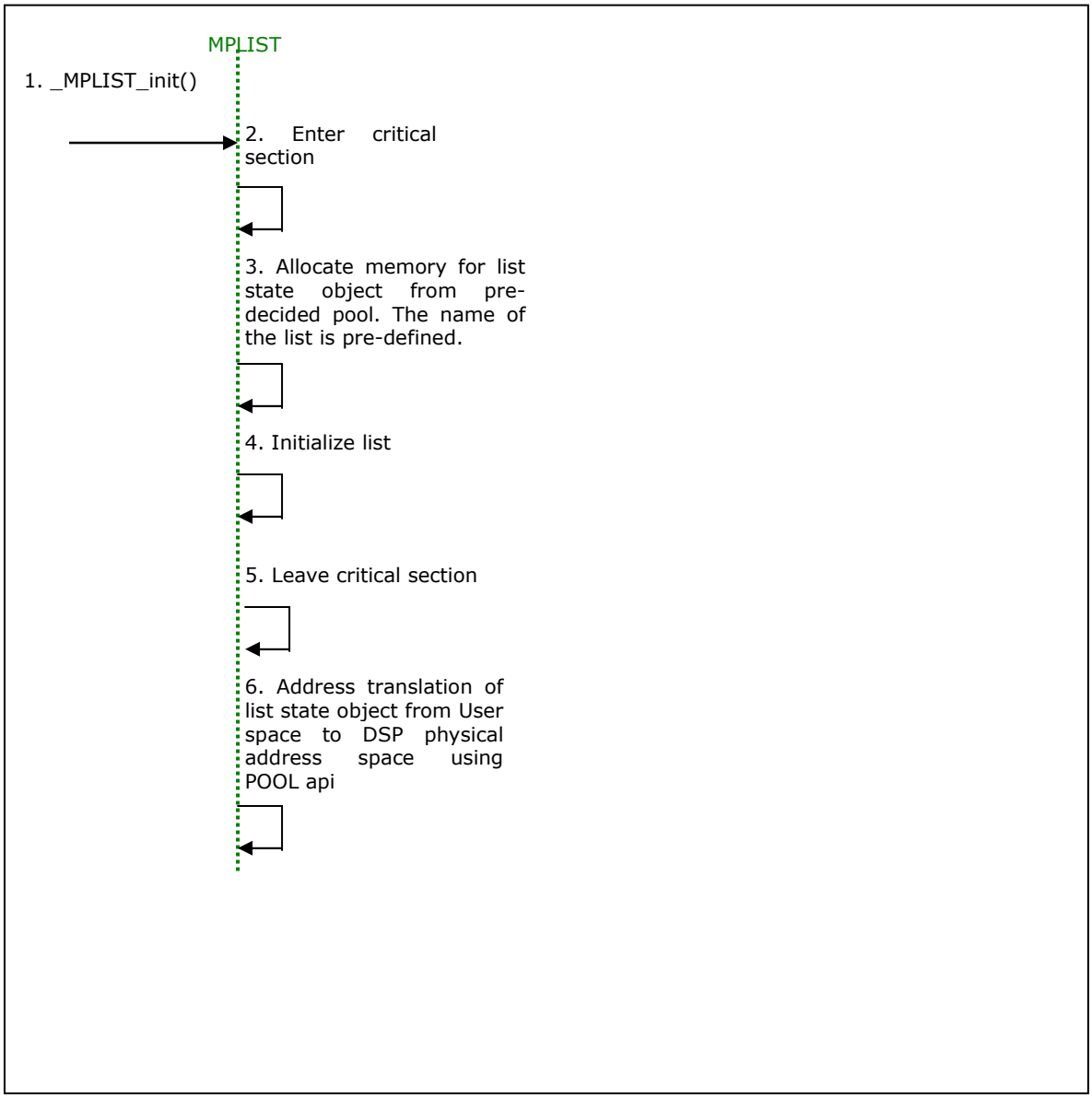

Figure 3. On the GPP: _MPLIST_init () control flow

6.2 MPLIST_create ()

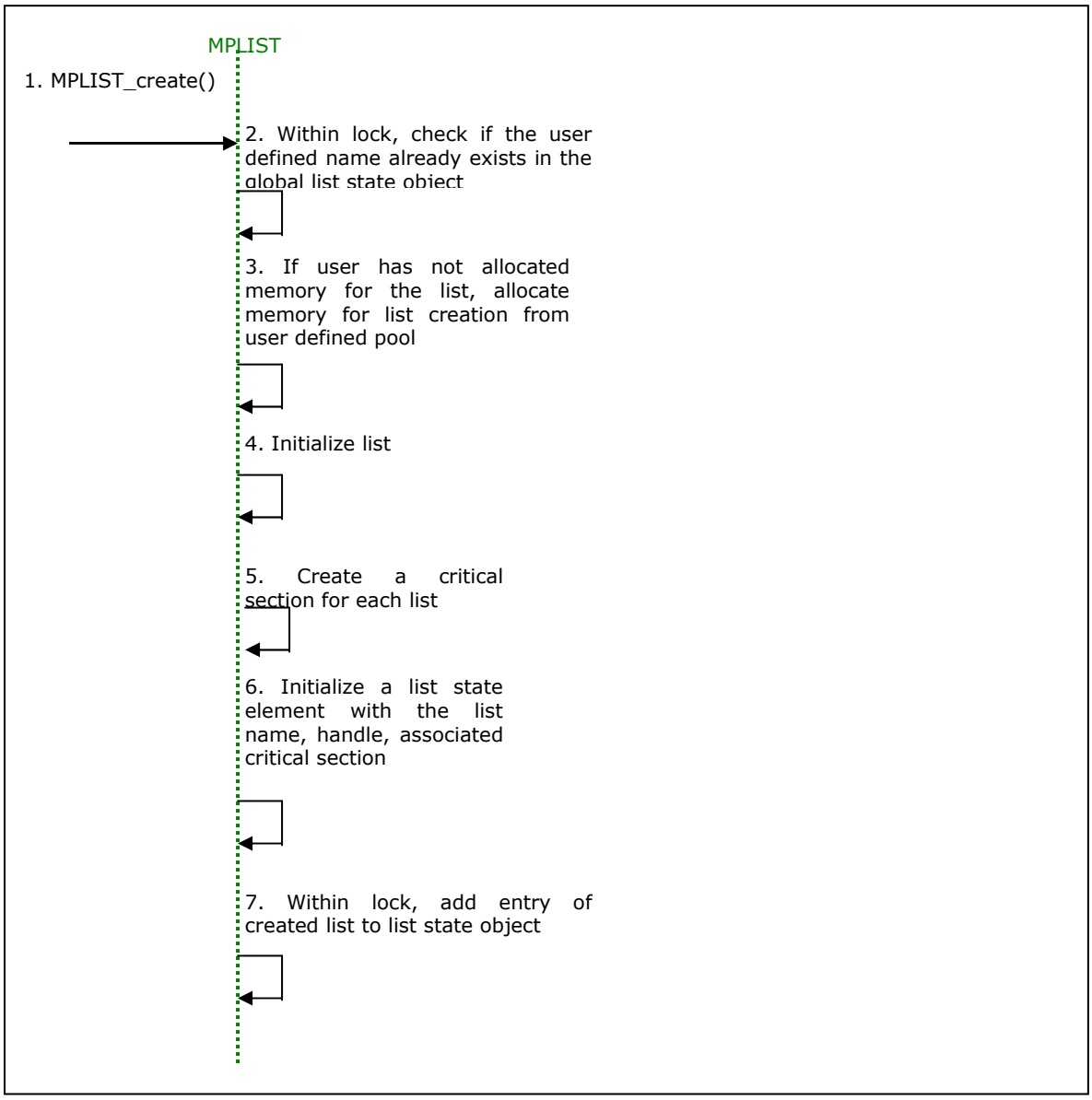

Figure 4. On the GPP: MPLIST_create () control flow

6.3 MPLIST_delete ()

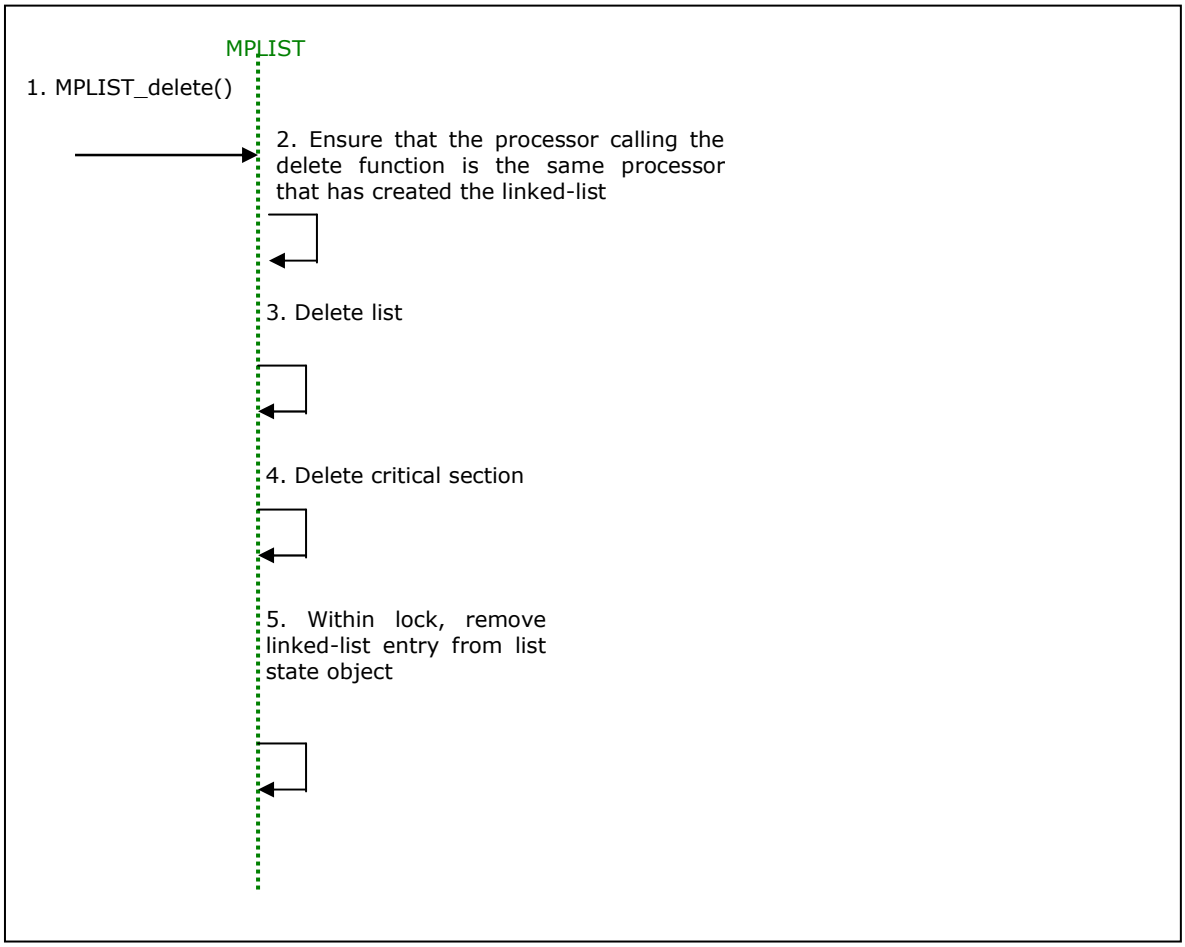

Figure 5. On the GPP: MPLIST_delete () control flow

6.4 MPLIST_getHead ()

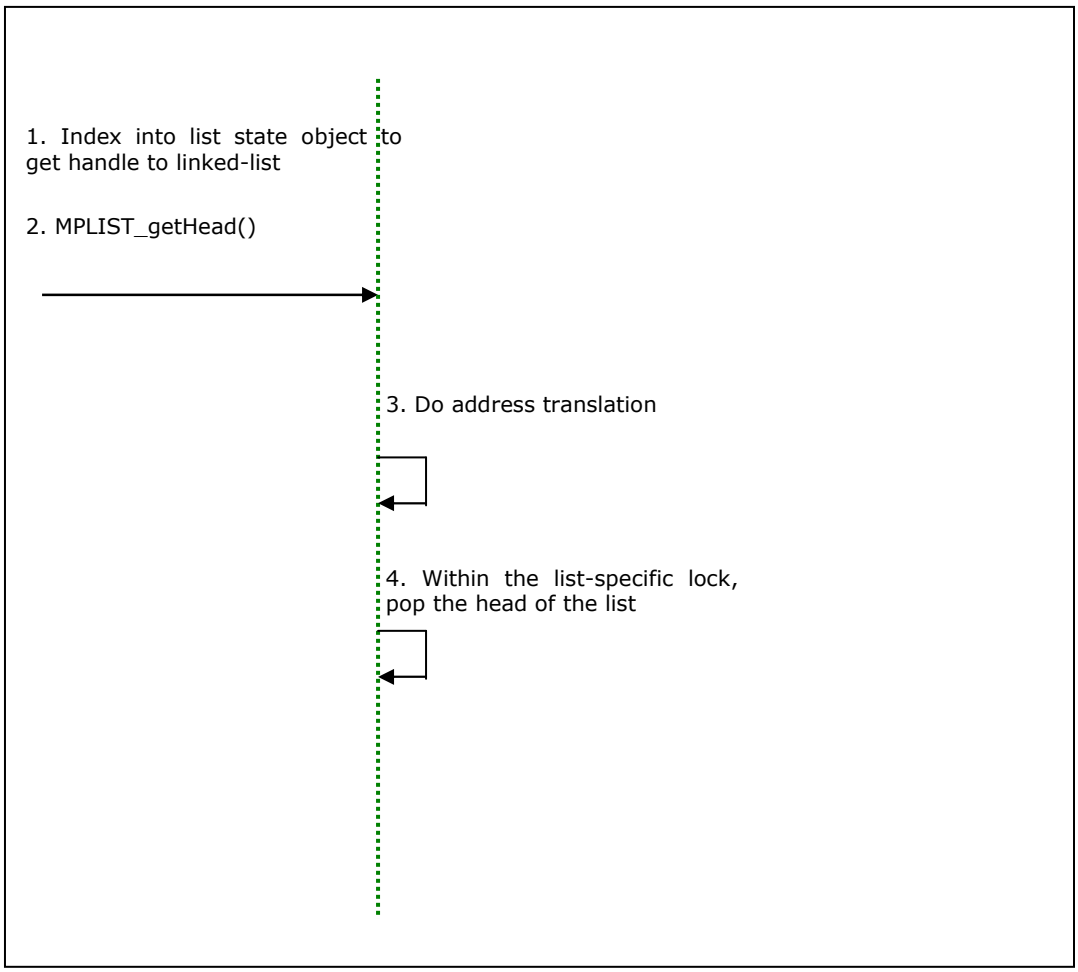

Figure 6. On the GPP: MPLIST_getHead () control flow

7 MPLIST

The MPLIST component has the same design on both the GPP and DSP sides. This section primarily refers to the GPP side design. However, the DSP-side design shall contain the same enumerations, structures, and API definitions, with minimal changes for different naming conventions on the GPP and DSP-sides.

7.1 GPP and DSP side low level design

7.1.1 Constants & Enumerations

None.

7.1.2 Typedefs & Data Structures

7.1.2.1 MPLIST_Attrs

This structure defines the MPLIST Creation Parameters.

Definition

```
typedef struct MPLIST_Attrs_tag {
    Uint16  poolId ;
} MPLIST_Attrs ;
```

Fields

`poolId`	Pool to be used to allocate memory for MPLIST data structures.

Comments

GPP-side: This structure defines the attributes required during opening of the MPLIST.

The attributes can contain the pool id specified by the user. This will determine from which pool the MPLIST_create function will allocate memory for the MPLIST data structures.

Constraints

None.

See Also

None.

7.1.2.2 MPLIST_Header

This structure defines an element of a list.

Definition

```
typedef struct MPLIST_Header_tag {
    struct Mplist_Header_tag *  next ;
    struct Mplist_Header_tag *  prev ;
} MPLIST_Header ;
```

Fields

`next`	Next node pointer.
`prev`	Previous node pointer.

Comments

GPP-side: This structure defines the element of a list. It contains a fixed size header along with pointers to the next and previous element of the list.

Constraints

None.

See Also

None.

7.1.2.3 MPLIST_List

This structure defines an MPLIST.

Definition

```
typedef struct MPLIST_List_tag {
    MPLIST_ElementHeader head ;
    ADD_PADDING  (padding, MPLIST_LIST_PADDING)
    MPCS_ShObj           mpcsObj ;
} MPLIST_List ;
```

Fields

`head`	Head of the list.
`padding`	Padding for cache-line alignment, if required.
`mpcsObj`	MPCS object used for protection of the list.

Comments

GPP-side: This structure defines the list. The field is the head of the list.

Constraints

None.

See Also

None.

7.1.2.4 MPLIST_Entry

This structure defines the MPLIST descriptor entry for each MPLIST in the system.

Definition

```
typedef struct MPLIST_Entry_tag {
    Uint16       ownerProcId ;
    Uint16       poolId ;
    Pvoid        phyListHandle  ;
    Char8        name [DSP_MAX_STRLEN] ;
    ADD_PADDING  (padding, MPLIST_ENTRY_PADDING)
} MPLIST_Entry ;
```

Fields

`ownerProcId`	Processor ID of the creator of the MpList.
`poolId`	Pool id to be used to allocate memory for all MPLIST data structures.
`phyListHandle`	Handle of the MPLIST in Physical (DSP) address space.
`name`	System wide unique identifier for the MPLIST.
`padding`	Padding for cache-line alignment, if required.

Comments

GPP-side:. This structure defines a single entry in the MPLIST entries table, which contains information about all MPLIST objects in the system.

Constraints

None.

See Also

None.

7.1.2.5 MPLIST_Ctrl

This structure defines the control structure required by the MPLIST component.

Definition

```
typedef struct MPLIST_Ctrl_tag {
    Uint32         isInitialized ;
    Uint32         dspId ;
    Uint32         maxEntries ;
    Uint32         ipsId ;
    Uint32         ipsEventNo ;
    MPLIST_Entry * dspAddrEntry ;
    ADD_PADDING    (padding, MPLIST_CTRL_PADDING)
    MPCS_ShObj     lockObj ;
} MPLIST_Ctrl ;
```

Fields

`isInitialized`	Flag to indicate if this region was initialized.
`dspId`	ID of the DSP with which the MPLIST region is shared.
`maxEntries`	Maximum number of MPLIST instances supported by the MPLIST.
`ipsId`	ID of the IPS to be used (if any). A value of -1 indicates that no IPS is required by the MPLIST.
`ipsEventNo`	IPS Event number associated with MPLIST (if any). A value of -1 indicates that no IPS is required by the MPLIST.
`dspAddrEntry`	Pointer to array in DSP address space of MPLIST objects that can be created.
`padding`	Padding for cache-line alignment, if required.
`lockObj`	Lock used to protect the shared MPLIST control region from multiple simultaneous accesses.

Comments

GPP-side:. This structure defines the control structure required by the MPLIST component. It contains information about all MPLIST objects shared between the GPP and a specific DSP.

Constraints

None.

See Also

None.

7.1.3 API Definition

7.1.3.1 MPLIST_create

This function creates a list in shared memory.

Syntax

```
DSP_STATUS MPLIST_create (ProcessorId    procId,
                          Char8 *        listName,
                          MPLIST_List *  mplistObj,
                          MPLIST_Attrs * attrs) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	Char8 *	listName
	List Name.	
IN OPT	MPLIST_List *	mplistObj
	List	
IN	MPLIST_attrs	attrs
	List Attributes.	

Return Value

DSP_SOK	This component has been successfully initialized.
DSP_EINVALIDARG	Invalid arguments.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_EALREADYEXISTS	The specified MPLIST name is already in use.
DSP_ERESOURCE	All MPLIST entries are currently in use.
DSP_EACCESSDENIED	The MPLIST component has not been initialized
DSP_ENOTFOUND	Information about specified POOL buffer was not available.
DSP_EFAIL	General failure.

Comments

GPP-side:

This function performs the following initialization:

- Within lock, check if the user defined name already exists in the global list state object.

- If user has not allocated memory for the list, allocate memory for list creation from user defined buffer pool. If user has already allocated memory for list creation then this function, then skip this step.
- Initialize the list.
- Create a critical section for each list
- Initialize a list state element with the list name, handle, associated critical section.
- Within lock, add entry of created list to list state object.

DSP-side:

The API definition is:

```
Int MPLIST_create
```

This DSP-side function performs similar initialization as GPP side.

Constraints

The processor which does list creation should be the processor which does the list deletion.

See Also

```
MPLIST_delete
```

7.1.3.2 MPLIST_delete

This function deletes a list from shared memory.

Syntax

```
DSP_STATUS MPLIST_delete (ProcessorId   procId,
                          Char8 *       listName) ;
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	Char8 *	listName
	List Name.	

Return Value

DSP_SOK	This component has been successfully deleted.
DSP_EINVALIDARG	Invalid arguments.
DSP_EMEMORY	Operation failed due to a memory error.
DSP_ENOTFOUND	The specified MPLIST is not present. Information about specified POOL buffer was not available.
DSP_ERESOURCE	All MPLIST entries are currently in use.
DSP_EACCESSDENIED	The MPLIST component has not been initialized.
DSP_EFAIL	General failure.

Comments

GPP-side:

This function performs the following function:

- Free memory for list.
- Removal of list related variables which could include a critical section.
- Removal of list entry in global control structure which contains all list entries.

DSP-side:

The API definition is:

```
Int MPLIST_delete
```

This DSP-side function performs similar function as GPP side.

Constraints

The processor which does list creation should be the processor which does the list deletion.

See Also

`MPLIST_create`

7.1.3.3 MPLIST_open

This function opens an MPLIST object specified by its name and gets a handle to the object. Every process that needs to use the MPLIST object must get a handle to the object by calling this API.

Syntax

```
DSP_STATUS MPLIST_open (ProcessorId     procId,
                        Char8 *         name,
                        MPLIST_Handle * mplistHandle
```

Arguments

IN	ProcessorId	procId
	Processor Identifier.	
IN	Char8 *	name
	List Name.	
OUT	MPLIST_Handle	listHandle
	Handle for List Operations.	

Return Value

DSP_SOK	This component has been successfully initialized.
DSP_EINVALIDARG	Invalid arguments.
DSP_ENOTFOUND	The specified MPLIST is not present. Information about specified POOL buffer was not available.
DSP_ERESOURCE	All MPLIST entries are currently in use.
DSP_EACCESSDENIED	The MPLIST component has not been initialized
DSP_EFAIL	General failure.

Comments

GPP-side:

This function performs the following initialization:

- Check if the user defined name already exists in the global list state object.
- Allocate memory for list handle in process space.
- Open the critical section lock handle.
- Fill handle with valid pointers to list, list entry and lock object

DSP-side:

The API definition is:

`Int MPLIST_open`

This DSP-side function performs similar initialization as GPP side.

Constraints

None.

See Also

`MPLIST_close`

7.1.3.4 MPLIST_close

This function closes the MPLIST associated with the process calling all list operations.

Syntax

```
DSP_STATUS MPLIST_close  (MPLIST_Handle mplistHandle) ;
```

Arguments

IN	MPLIST_Handle	listHandle

Handle for List Operations.

Return Value

DSP_SOK	This component has been successfully deleted.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_SFREE	Resource has been freed successfully
DSP_EACCESSDENIED	The MPLIST component has not been initialized

Comments

GPP-side:

This function performs the following function:

- Clearing of list handle contents
- Free memory for list handle.

DSP-side:

The API definition is:

```
Int MPLIST_close
```

This DSP-side function performs similar function as GPP side.

Constraints

None.

See Also

```
MPLIST_open
```

7.1.3.5 MPLIST_isEmpty

This function checks whether the list is empty.

Syntax

```
Bool MPLIST_isEmpty (MPLIST_Handle listHandle) ;
```

Arguments

IN	MPLIST_Handle	listHandle

Handle for List Operations.

Return Value

TRUE	The list is empty.
FALSE	General failure.

Comments

GPP-side:

This function performs the following function:

- Checks if given list is empty.

DSP-side:

The API definition is:

```
Bool MPLIST_isEmpty
```

This DSP-side function performs similar function as GPP.

Constraints

None.

See Also

None.

7.1.3.6 MPLIST_getHead

This function pops the head pointer from the list.

Syntax

```
DSP_STATUS MPLIST_getHead (MPLIST_Handle listHandle,
                           MPLIST_Elem     * listElement) ;
```

Arguments

IN	MPLIST_Handle	listHandle
	Handle for List Operations.	
OUT	MPLIST_Elem	listElement
	List Element.	

Return Value

DSP_SOK	The head of the list is returned successfully.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_ENOTFOUND	Information about specified POOL buffer was not available.

Comments

GPP-side:

This function performs the following function:

- Pops the head of the list into listElelement.

DSP-side:

The API definition is:

```
Int MPLIST_getHead
```

This DSP-side function performs similar function as the GPP.

Constraints

None.

See Also

None.

7.1.3.7 MPLIST_putTail

This function adds the specified element to the tail of the list.

Syntax

```
DSP_STATUS MPLIST_putTail (MPLIST_Handle listHandle,
                           MPLIST_Elem       listElement) ;
```

Arguments

IN	MPLIST_Handle	listHandle
	Handle for List Operations.	
IN	MPLIST_Elem	listElement
	List Element.	

Return Value

DSP_SOK	This component has been added to the tail of the list.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_ENOTFOUND	Information about specified POOL buffer was not available.

Comments

GPP-side:

This function performs the following function:

- Adds the specified element to the tail of the list.

DSP-side:

The API definition is:

```
Int MPLIST_putTail
```

This DSP-side function performs similar function as GPP side.

Constraints

None.

See Also

None.

7.1.3.8 MPLIST_removeElement

This function removes (unlinks) the given element from the list.

Syntax

```
DSP_STATUS MPLIST_removeElement (MPLIST_Handle listHandle,
                                 MPLIST_Elem       listElement) ;
```

Arguments

IN	MPLIST_Handle	listHandle
	Handle for List Operations.	
IN	MPLIST_Elem	listElement
	List Element.	

Return Value

DSP_SOK	This list element has been successfully deleted.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_ENOTFOUND	Information about specified POOL buffer was not available. The specified buffer was not present in POOL.

Comments

GPP-side:

This function performs the following function:

- Removes (unlinks) the given element from the list.

DSP-side:

The API definition is:

```
Int MPLIST_removeElement
```

This DSP-side function performs similar function as GPP side.

Constraints

None.

See Also

None.

7.1.3.9 _MPLIST_init

This function creates the global list state object.

Syntax

```
DSP_STATUS _MPLIST_init   (ProcessorId        procId) ;
```

Arguments

IN	ProcessorId	procId

Processor Identifier.

Return Value

DSP_SOK	The global list state object has been successfully created.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.

Comments

GPP-side:

This function performs the following initialization:

- Creates the list state object
- Initializes the list state object
- The name of this list is known to both GPP and DSP which enables access by all writers.

DSP-side:

The API definition is:

```
Int _MPLIST_init
```

This DSP-side function performs similar initialization as GPP side.

Constraints

None.

See Also

_MPLIST_exit

7.1.3.10 _MPLIST_exit

This function finalizes the MPLIST component.

Syntax

```
DSP_STATUS _MPLIST_exit   (ProcessorId        procId) ;
```

Arguments

IN	ProcessorId	procId

Processor Identifier.

Return Value

DSP_SOK	The global list state object has been successfully deleted.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_EMEMORY	Operation failed due to memory error.

Comments

GPP-side:

This function performs the following initialization:

- Frees the memory required to store information about each list present in the system
- Frees the memory required to store the list object.

DSP-side:

The API definition is:

```
Int _MPLIST_exit
```

This DSP-side function performs similar initialization as GPP side.

Constraints

None.

See Also

_MPLIST_init

7.1.3.11 MPLIST_first

This function returns a pointer to the first element of the list.

Syntax

```
DSP_STATUS MPLIST_first(MPLIST_Handle   listHandle,
                        MPLIST_Elem   * listElement
```

Arguments

IN	MPLIST_Handle	listHandle
	Handle for List Operations.	
OUT	MPLIST_Elem	listElement
	List Element.	

Return Value

DSP_SOK	The first element of the list has been successfully returned.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_ENOTFOUND	Information about specified POOL buffer was not available. The specified buffer was not present in POOL.

Comments

GPP-side:

This function performs the following function:

- Returns a pointer to the first element of the list, or NULL if the list is empty.

DSP-side:

The API definition is:

```
Int MPLIST_first
```

This DSP-side function performs similar function as GPP side.

Constraints

None.

See Also

None.

7.1.3.12 MPLIST_next

This function returns a pointer to the next element of the list.

Syntax

```
DSP_STATUS MPLIST_next(MPLIST_Handle listHandle,
                       MPLIST_Elem   currentElement,
                       MPLIST_Elem * nextElement)
```

Arguments

IN	MPLIST_Handle	listHandle
	Handle for List Operations.	
IN	MPLIST_Elem	currentElement
	List Element.	
OUT	MPLIST_Elem	nextElement
	List Element.	

Return Value

DSP_SOK	The next element of the list has been successfully returned.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_ENOTFOUND	Information about specified POOL buffer was not available. The specified buffer was not present in POOL.

Comments

GPP-side:

This function performs the following function:

- Returns a pointer to the next element of the list, or NULL if the list is empty or the next element is the head of the list.

DSP-side:

The API definition is:

```
Int MPLIST_next
```

This DSP-side function performs similar function as GPP side.

Constraints

None.

See Also

None.

7.1.3.13 MPLIST_insertBefore

This function inserts an element before the existing element in the list.

Syntax

```
DSP_STATUS MPLIST_insertBefore(MPLIST_Handle listHandle,
                               MPLIST_Elem   insertElement,
                               MPLIST_Elem   existingElement)
```

Arguments

IN	MPLIST_Handle	listHandle

Handle for List Operations.

IN	MPLIST_Elem	insertElement

List Element to be inserted.

OUT	MPLIST_Elem	existingElement

Existing List Element.

Return Value

DSP_SOK	The next element of the list has been successfully returned.
DSP_EFAIL	General failure.
DSP_EINVALIDARG	Invalid arguments.
DSP_ENOTFOUND	Information about specified POOL buffer was not available. The specified buffer was not present in POOL.

Comments

GPP-side:

This function performs the following function:

- Inserts a new element before an existing element in the list.

DSP-side:

The API definition is:

```
Int MPLIST_insertBefore
```

This DSP-side function performs similar function as GPP side.

Constraints

None.

See Also

None.