DSP/BIOS™ LINK

OS ADAPTATION LAYER FOR PrOS

LNK 096 DES

Version 1.30

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

1.1 Purpose and Scope

This document describes the overall design and architecture of the OS Adaptation Layer (OSAL) of DSP/BIOS™ LINK for PrOS.

It lists the interfaces exposed by the OSAL and also describes the overall design for implementation of these interfaces.

The document may not reflect all the return values that a function may return.

1.2 Terms and Abbreviations

Term	Definition or explanation
OSAL	OS Adaptation Layer

1.3 References

1.	LNK 002 ARC	DSP/BIOS Link High Level Architecture Version 1.02 dated JUL 15, 2003
2.	LNK 003 REQ	DSP/BIOS Link OS Adaptation Layer Requirements Version 1.00 dated JUN 12, 2002

1.4 Overview

OSAL provides an abstraction from the basic services of the underlying OS to the sub-components of DSP/BIOS™ LINK.

Since the OSAL modules interface directly with the underlying OS, it provides portability to any component built on top of it, as long as the interfaces documented in this document are ported to the target OS.

Its intended audience is the design and implementation team of DSP/BIOS™ LINK.

2 CFG

This component provides the functionality to specify the configuration parameters that the users of DSP/BIOS LINK may want to change according to the target system

2.1 Resources Available

None.

2.2 Dependencies

2.2.1 Subordinates

None.

2.2.2 Preconditions

None.

2.3 Description

The configuration data is stored in structures. The CFG subcomponent provides the required services to access these structures using predefined KEYS.

2.4 Typedefs and Data Structures

2.4.1 CFG_Driver

Driver configuration structure

Definition

```
typedef struct CFG_Driver_tag {
    Char8  driverName [CFG_MAX_STRLEN]    ;
    Uint32 components                     ;
#if defined (CHNL_COMPONENT)
    Uint32 queueLength                    ;
#endif /* if defined (CHNL_COMPONENT) */
    Uint32 numDsps                        ;
    Uint32 memTables                      ;
#if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT)
    Uint32 numIpsTables                   ;
    Uint32 numPools                       ;
#endif /* if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT) */
#if defined (CHNL_COMPONENT)
    Uint32 numDataTables                  ;
#endif /* if defined (CHNL_COMPONENT) */
#if defined (MSGQ_COMPONENT)
    Uint32 maxMsgqs                       ;
    Uint32 numMqts                        ;
#endif /* if defined (MSGQ_COMPONENT) */
#if defined (DDSP_PROBE)
    Uint32 probeRtcId                     ;
    Uint32 probeRtcIntId                  ;
#endif /* if defined (DDSP_PROBE) */
} CFG_Driver ;
```

Fields

`driverName`	Name of the driver
`components`	Number of components of driver
`queueLength`	Maximum number of buffer queues. Included only if CHNL component is included.
`numDsps`	Number of DSPs in the current configuration. Included only if CHNL component is included.
`memTables`	Number of MEM tables in the current configuration. Included only if CHNL component is included.
`numIpsTables`	Number of Inter Processor Signalling components tables in the current configuration. Included only if CHNL component/MSGQ component is included.
`numPools`	Number of memory pools supported by the current configuration. Included only if CHNL component/MSGQ component is included.

numDataTables	Number of data transfer drivers tables supported in current configuration. Included only if CHNL component is included.
maxMsgqs	Maximum number of MSGQs supported in this configuration. Included only if MSGQ component is included.
numMqts	Number of Transports supported in current configuration. Included only if MSGQ component is included.
probeRtcId	The id of the timer to be used for PROBE. This field is defined only when probe component is enabled.
probeRtcIntId	The ISR id of the timer to be used for PROBE. This field is defined only when probe component is enabled.

Comments

This structure defines general driver related configuration items. The fields defined within MSGQ_COMPONENT are only required when messaging is scaled in. The fields defined within CHNL_COMPONENT are only required when data transfer using channels is scaled in.

2.4.2 CFG_Gpp

It specifies the general configuration parameters for the GPP side.

Definition

```
typedef struct CFG_Gpp_tag {
    Char8   gppName [CFG_MAX_STRLEN] ;
} CFG_Gpp ;
```

Fields

gppName	Name of GPP Processor

Comments

None.

2.4.3 CFG_Dsp

It specifies the general configuration parameters for the DSP processor.

Definition

```
typedef struct CFG_Dsp_tag {
    Char8   dspName    [CFG_MAX_STRLEN] ;
    Uint32  dspArch          ;
    Pvoid   interface        ;
    Pvoid   loaderInterface  ;
    Uint32  autoStart        ;
    Char8   execName   [CFG_MAX_STRLEN] ;
    Uint32  resetVector      ;
    Uint32  maduSize         ;
    Uint32  endian           ;
    Uint32  memEntries       ;
    Uint32  memTable         ;
#if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT)
    Uint32  linkdrvId        ;
#endif /* if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT) */
#if defined (CHNL_COMPONENT)
    Uint32  numDataDrivers   ;
    Uint32  dataTableId      ;
#endif /* if defined (CHNL_COMPONENT) */
#if defined (MSGQ_COMPONENT)
    Uint32  mqtId            ;
#endif /* if defined (MSGQ_COMPONENT) */
} CFG_Dsp ;
```

Fields

`dspName`	Name of DSP Processor
`dspArch`	Architecture of the DSP.
`interface`	Function pointer interface for accessing the DSP.
`loaderInterface`	Function pointer interface for accessing the loader.
`autoStart`	AutoStart flag.
`execName`	Name of the default DSP executable.
`resetVector`	Address of reset vector of DSP.
`maduSize`	Minimum addressable unit on the DSP.
`endian`	Endian info of DSP.
`memEntries`	Number of entries in MEM table.
`mmuTable`	Table number of the MEM entries for this DSP.
`linkdrvId`	Link Driver table identifier for this DSP.
`numDataDrivers`	Size of the link table.

`dataTableId`	Table number of the link(s) toward this DSP.
`mqtId`	The id of the MQT to be used for this DSP.

Comments

Base of the keys to fetch DSP related information from the configuration database.

2.4.4 CFG_Link

Base of the keys to fetch link related information from the configuration database.

Definition

```
#if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT)
typedef struct CFG_LinkDrv_tag {
    Char8   linkDrvName [CFG_MAX_STRLEN] ;
    Char8   abbr        [CFG_MAX_STRLEN] ;
    Pvoid   interface     ;
    Uint32  memEntry      ;
    Uint32  size          ;
    Uint32  numIpsEntries ;
    Uint32  ipsTableId    ;
} CFG_LinkDrv ;
#endif /* if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT) */
```

Fields

`linkName`	Name of Link.
`abbr`	Abbreviation of the link name.
`interface`	Function pointer interface for the physical link driver.
`memEntry`	MEM Entry for the memory area used by this physical link configuration information.
`size`	Size of memory area for the physical link configuration information.
`numIpsEntries`	Number of IPS table entries for this link driver.
`ipsTableId`	IPS table id

Comments

It specifies the Link configuration parameters. It is defined only when either MSGQ or Channel component is included.

2.4.5 CFG_MmuEntry

Defines an entry in the MMU table.

Definition

```
typedef struct CFG_MemEntry_tag {
    Uint32  entry            ;
    Char8   abbr [CFG_MAX_STRLEN] ;
    Uint32  dspVirtualAddress ;
    Uint32  physicalAddress   ;
    Uint32  size              ;
    Uint32  mapInGpp          ;
} CFG_MemEntry ;
```

Fields

`entry`	Entry number
`abbr`	Abbreviation to be used for generating base address for this memory block.
`dspVirtualAddress`	virtual address field of entry.
`physicalAddress`	physical address field of entry.
`size`	Size field of entry.
`mapInGpp`	Flag indicating whether DSP address is mapped to GPP address space.

Comments

It specifies an entry in the MMU table.

2.4.6 CFG_Pools

This structure defines Configuration information for buffer pools from which buffers for use with LINK can be allocated through the POOL interface(s). The buffers can be used for both - message as well as data transfer.

Definition

```
#if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT)
typedef struct CFG_Pools_tag {
    Char8   poolName [CFG_MAX_STRLEN] ;
    Char8   abbr     [CFG_MAX_STRLEN] ;
    Pvoid   initialize ;
    Pvoid   finalize   ;
    Pvoid   interface  ;
    Uint32  dspId      ;
    Uint32  memEntry   ;
    Uint32  poolSize   ;
    Uint32  arg1       ;
    Uint32  arg2       ;
} CFG_Pools ;
#endif /* if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT) */
```

Fields

`poolName`	Name of the pool.
`abbr`	Abbreviated name for generating macros.
`initialize`	Initialization function for the buffer pool.
`finalize`	Finalization function for the buffer pool.
`interface`	Function interface table for the buffer pool.
`dspId`	DSP Identifier for the pool.
`memEntry`	MEM Entry for the memory area used by this buffer pool. This field is optional and can be specified as -1 in case a MEM entry is not required for the buffer pool (e.g. in case of PCPY Messaging).
`poolSize`	Size of the buffer pool.
`arg1`	Optional pool specific argument.
`arg2`	Optional pool specific argument.

Comments

This structure is defined only if `MSGQ_COMPONENT/CHNL_COMPONENT` is enabled.

2.4.7 CFG_Mqt

This structure defines the MQT configuration structure.

Definition

```
#if defined (MSGQ_COMPONENT)
typedef struct CFG_Mqt_tag {
    Char8   mqtName [CFG_MAX_STRLEN] ;
    Char8   abbr    [CFG_MAX_STRLEN] ;
    Pvoid   interface  ;
    Uint32  memEntryId ;
    Uint32  maxMsgSize ;
    Uint32  size       ;
    Uint32  arg1      ;
    Uint32  arg2      ;
} CFG_Mqt ;
#endif /* if defined (MSGQ_COMPONENT) */
```

Fields

`mqtName`	Name of the MQT. For debugging purposes only.
`Abbr`	Abbreviation of the MQT name.
`interface`	ID of the link used by this MQT.
`memEntryId`	Function pointer interface to access the functions for this MQT.
`maxMsgSize`	MEM entry id for the memory area reserved for this MQT.
`size`	Size of memory area for this MQT.
`arg1`	Optional argument 1 for this MQT.
`arg2`	Optional argument 2 for this MQT.

Comments

This structure is defined only if `MSGQ_COMPONENT` is enabled.

2.4.8 CFG_DataDrv

Data streaming driver configuration structure

Definition

```
#if defined (CHNL_COMPONENT)
typedef struct CFG_DataDrv_tag {
    Char8   name [CFG_MAX_STRLEN] ;
    Char8   abbr [CFG_MAX_STRLEN] ;
    Uint32  baseChnlId     ;
    Uint32  numChannels    ;
    Uint32  maxBufSize     ;
    Pvoid   interface      ;
    Uint32  memEntryId     ;
    Uint32  poolId         ;
    Uint32  size           ;
    Uint32  arg1           ;
    Uint32  arg2           ;
} CFG_DataDrv ;
#endif /* if defined (CHNL_COMPONENT) */
```

Fields

`name`	Name of the MQT. For debugging purposes only.
`abbr`	Abbreviation of the link name.
`baseChnlId`	Base channel ID for this link.
`numChannels`	Number of channels for this link.
`maxBufSize`	Maximum size of data buffer on this link.
`interface`	Interface function table address.
`memEntryId`	MEM entry for the memory area for data streaming driver.
`poolId`	Identifier for the pool from where buffers are allocated.
`size`	Size of the area for data streaming driver.
`arg1`	Link specific argument 1. The significance of this arg is specific to a link driver.
`arg2`	Link specific argument 2. The significance of this argument is specific to a link driver.

Comments

This structure is defined only if CHNL_COMPONENT is enabled.

2.4.9 CFG_Ips

Configuration information for the Interprocessor Signaling Component

Definition

```
#if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT)
typedef struct CFG_Ips_tag {
    Char8   ipsName [CFG_MAX_STRLEN] ;
    Char8   abbr [CFG_MAX_STRLEN] ;
    Pvoid   initialize   ;
    Pvoid   finalize     ;
    Uint32  queuePerChnl ;
    Uint32  irpSize      ;
    Uint32  memEntry     ;
    Uint32  size         ;
    Uint32  arg1         ;
    Uint32  arg2         ;
} CFG_Ips ;
#endif /* if defined (CHNL_COMPONENT) || defined (MSGQ_COMPONENT) */
```

Fields

`ipsName`	Name of the IPS component.
`abbr`	Abbreviated name for generating macros.
`initialize`	Function pointer for the initialization function.
`finalize`	Function pointer for the finalization function.
`queuePerChnl`	Number of IO requests to be queued within IPS for data transfer.
`irpSize`	Size of each IRP packet.
`memEntry`	MEM entry for the memory information used for this IPS component.
`size`	Size of memory area configured for the IPS component.
`arg1`	IPS specific optional argument.
`arg2`	IPS specific optional argument.

Comments

This structure is defined only if `MSGQ_COMPONENT/CHNL_COMPONENT` is enabled.

2.5 API Definition

2.5.1 CFG_Initialize

This function initializes CFG sub-component.

Syntax

```
DSP_STATUS CFG_Initialize () ;
```

Arguments

None.

Return Values

DSP_SOK	Component initialized successfully.

Comments

Current implementation does not have any functionality in this function.

Constraints

None.

See Also

CFG_Finalize

2.5.2 CFG_Finalize

This function provides an interface to exit from this sub-component.

Syntax

```
DSP_STATUS CFG_Finalize () ;
```

Arguments

None.

Return Values

DSP_SOK	Component finalized successfully.

Comments

Current implementation does not have any functionality in this function. After this function call, CFG sub-component must not be used.

Constraints

Module must have been initialized.

See Also

CFG_Initialize

2.5.3 CFG_GetRecord

Gets a particular configuration record in a structure.

Syntax

```
DSP_STATUS CFG_GetRecord (Uint32 key, Uint32 id, Void * record) ;
```

Arguments

IN	Uint32	key

Key for the configuration.

IN	Uint32	id

Record Id for which configuration is requested

OUT	Void *	record

Place where to copy the required configuration data

Return Values

DSP_SOK	Operation Successful.
DSP_EFAIL	Operation failed. Requested configuration data not found.
DSP_EINVALIDARG	Invalid key specified.

Comments

None.

Constraints

record must be a valid pointer.

See Also

CFG_GetNumValue, CFG_GetStrValue

2.5.4 CFG_GetNumValue

Gets a particular configuration parameter as a numeric value.

Syntax

```
DSP_STATUS CFG_GetNumValue (Uint32 key, Uint32 id, Uint32 * value) ;
```

Arguments

IN	Uint32	key

Key for the configuration.

IN	Uint32	id

Id of record in which to look for the requested value

OUT	Uint32 *	value

Place where to copy the required configuration data

Return Values

DSP_SOK	Operation Successful.
DSP_EFAIL	Operation failed. Requested configuration data not found.
DSP_EINVALIDARG	Invalid key specified.

Comments

None.

Constraints

value must be a valid pointer.

See Also

CFG_GetRecord, CFG_GetStrValue

2.5.5 CFG_GetStrValue

Gets a particular configuration parameter as a string.

Syntax

```
DSP_STATUS CFG_GetStrValue (Uint32 key, Uint32 id, Pstr string) ;
```

Arguments

IN	Uint32	key

Key for the configuration.

IN	Uint32	id

Id of record in which to look for the requested value

OUT	Pstr	string

Place where to copy the required configuration data

Return Values

DSP_SOK	Operation Successful.
DSP_EFAIL	Operation failed. Requested configuration data not found.
DSP_EINVALIDARG	Invalid key specified.

Comments

None.

Constraints

string must be a valid pointer.

See Also

CFG_GetRecord, CFG_GetNumValue

3 DPC

This component provides the services of a Deferred Procedure Call. It allows execution of non time-critical code to be postponed to a later point of time.

3.1 Resources Available

PrOS does not provide support for deferred function calls. DPCs are implemented using task with highest priority after kernel, interrupts and exceptions. These tasks run at priority 4.

3.2 Dependencies

3.2.1 Subordinates

`SYNC, MEM, TRC`

3.2.2 Preconditions

None.

3.3 Description

DPC object contains an instance of `task` object. A `DpcObject` is associated with an interrupt. When the interrupt occurs, it calls `DPC_Schedule ()` with the `DpcObject` as the argument. `DPC_Schedule ()` schedules its associated `task`. When the `task` is scheduled to run by PrOS kernel, the `DPC_CallBack ()` function is invoke. DPC_CALLBACK in turn calls the function pointed by `UserDPCFn` with reference data pointed by `ParamData`.

The `Dpcs` array in `DPC_DpcTasksTable` structure is a placeholder that stores all DPC objects. The `UsedDPCs` field in this structure is a bit mask that keeps track of `DPCs` that are in use.

3.4 Typedefs and Data Structures

Definition

```
typedef Void (*FnDpcProc) (Pvoid refData) ;
```

Comments

This is the function signature for a user supplied DPC function for DSP/BIOS Link.

3.4.1 DpcObject

This object stores information related to a deferred procedure call.

Definition

```
struct DpcObject_tag {
    Uint32    signature    ;
    Uint32    index        ;
    Pvoid     paramData    ;
    FnDpcProc userDPCFn    ;
    ID        tskId        ;
    Bool      isEnabled    ;
    Bool      isCancelled  ;
} ;
```

Fields

`signature`	Signature identifying the DPC objects. Is the literal string "DPC_"
`index`	Index of the DPC object
`paramData`	Parameter to be passed to the deferred function call
`userDPCFunc`	Pointer to the user supplied function
`tskId`	ID number of task.
`isEnabled`	Flag, which tells the state of the DPC (Enabled or Disabled).
`isCancelled`	Flag, which tells the state of the DPC (if it was cancelled or not).

Comments

None.

Constraints

None.

See Also

`DpcTasksMap`

3.4.2 DpcTasksMap

This structure defines the association between DpcObjects and their corresponding task. It also contains a bitmap for tracking used Dpc objects.

Definition

```
typedef struct DpcTasksMap_tag {
    Uint32     usedDPCs         ;
    DpcObject  dpcs  [MAX_DPC]  ;
    T_CTSK     cTasks [MAX_DPC]  ;
} DpcTasksMap ;
```

Fields

usedDPCs	A Bitmap to keep track of DPCs that have been currently allocated and are used
dpcs	An Array to hold MAX_DPC number of DPC objects
cTasks	task objects to be used in conjunction with DpcObjects

Comments

This structure is a placeholder for all DPC objects and their associated usage information.

Constraints

None.

See Also

DpcObject
DPC_Initialize
DPC_Create
DPC_Destroy

3.5 API Definition

3.5.1 DPC_Initialize

This function initializes the DPC module. It initializes the global area (`DpcTasksMap` structure) for holding all the DPC objects and marks the UsedDPCs bitmap to indicate that no DPCs are currently in use.

Syntax

```
DSP_STATUS DPC_Initialize () ;
```

Arguments

None.

Return Values

`DSP_SOK`	Successful initialization.
`DSP_EMEMORY`	Out of memory error.

Comments

None.

Constraints

None.

See Also

```
DpcTasksMap
DPC_Create
DPC_Finalize
```

3.5.2 DPC_Finalize

This function releases all resources held by this sub-component.

Syntax

```
DSP_STATUS DPC_Finalize () ;
```

Arguments

None.

Return Values

`DSP_SOK`	Successful initialization.
`DSP_EMEMORY`	Out of memory error.
`DSP_EFAIL`	General error from GPP-OS.

Comments

During this function call it kills the tasks associated with any DPCs that may be in use.

Constraints

The sub-component must have been initialized.

See Also

DPC_Initialize

3.5.3 DPC_Create

Creates a DPC object and returns it after populating relevant fields.

Syntax

```
DSP_STATUS DPC_Create (FnDpcProc      userDPCFn,
                       Pvoid          dpcArgs,
                       DpcObject **   dpcObj) ;
```

Arguments

IN	FnDpcProc	userDPCFn
	User specified DPC function	
IN	Pvoid	dpcArgs
	Arguments to the user specified DPC function	
OUT	DpcObject **	dpcObj
	Pointer to the DPC object to be created	

Return Values

DSP_SOK	Successfull creation of DPC Object.
DSP_EINVALIDARG	Invalid parameters.
DSP_ERESOURCE	Maximum allowable number of DPCs created.

Comments

A call to DPC_Create() results in it reserving a DpcObject from the array, DPC_DpcTasksTable->dpcs. The corresponding index bit in DPC_DpcTasksTable->usedDPCs is set. The callback is then set to DPC_Callback ().

Constraints

This sub-component must be initialized.

userDPCFn must be a valid function.

dpcObj must be a valid DPC object.

See Also

DPC_Initialize
DPC_Schedule
DPC_Cancel
DPC_Callback
DPC_Delete

3.5.4 DPC_Delete

This function releases all resources associated with a DPC Object.

Syntax

```
DSP_STATUS DPC_Delete (DpcObject * dpcObj) ;
```

Arguments

IN	DpcObject *	dpcObj

DPC Object to be destroyed.

Return Values

DSP_SOK	Successful initialization.
DSP_EMEMORY	Memory error.
DSP_EFAIL	General error from GPP-OS.
DSP_EINVALIDARG	Invalid dpcObj parameter.
DSP_EPOINTER	Invalid handle to DpcObject.

Comments

This function kills the task associated with DpcObject, effectively canceling all pending calls to DPC. It also resets corresponding bit in DPC_DpcTasksTable->usedDPCs bitmask to indicate that the DPC is no longer used.

Constraints

This component must be initialized.

dpcObj must be a valid DPC object.

See Also

DPC_Create
DPC_Cancel

3.5.5 DPC_Cancel

This function cancels all pending calls to a DPC that were scheduled by DPC_Schedule () and have not yet been completed.

Syntax

```
DSP_STATUS DPC_Cancel (DpcObject * dpcObj) ;
```

Arguments

IN	DpcObject *	dpcObj

DPC Object to be cancelled.

Return Values

DSP_SOK	Successfully cancelled the DPC

DSP_EPOINTER	Invalid handle to DpcObject.
DSP_EINVALIDARG	Invalid DpcObject index.
DSP_EMEMORY	Memory error.
DSP_EFAIL	General error from GPP-OS.

Comments

This function sets the number of activation request for the DPC task to zero. By deleting and then re-creating the task.

Constraints

This sub-component must be initialized.

dpcObj must be a valid DPC object.

See Also

DPC_Initialize
DPC_Create
DPC_Schedule

3.5.6 DPC_Schedule

Schedules the user-defined function associated with dpcObj to be invoked at a later point of time.

Syntax

```
DSP_STATUS DPC_Schedule (DpcObject * dpcObj) ;
```

Arguments

IN DpcObject * dpcObj

DPC_Object to be scheduled.

Return Values

DSP_SOK	Successfully scheduled the DPC.
DSP_EPOINTER	Invalid handle to DpcObject.
DSP_EINVALIDARG	Invalid DpcObject index.
DSP_EFAIL	General error from GPP-OS.

Comments

This function calls act_tsk (). The DPC tasks are at a priority higher than any user task, so the DPC task start to execute immediately.

Constraints

This sub-component must be initialized.

dpcObj must be a valid DPC object.

See Also

DPC_Callback
DPC_Create

3.5.7 DPC_Debug

This function is used to print the current status of DPC objects in the system.

Syntax

```
Void DPC_Debug () ;
```

Arguments

None.

Return Values

None.

Comments

None.

Constraints

This function can only be used in debug builds.

See Also

None.

3.5.8 DPC_Callback

The OS kernel calls this function when a DPC is scheduled to run.

Syntax

```
void DPC_Callback (unsigned long index) ;
```

Arguments

IN unsigned log index

Indicates in index in DPC object table.

Return Values

None.

Comments

This function checks if there are any pending calls to the DPC and invokes the user specified function in a loop to service all pending calls.

Constraints

This function is called by PrOS Kernel as `task` entry point.

See Also

DPC_Create
DPC_Schedule

3.5.9 FnDpcProc

Function prototype for DPC function. The user defined function that is to be invoked as a DPC should conform to this signature.

Syntax

```
Void (*FnDpcProc) (Pvoid refData)
```

Arguments

IN	Pvoid	RefData

Argument to be passed to DPC call.

Return Values

None.

Comments

None.

Constraints

None.

See Also

DPC_Callback
DPC_Create

4 ISR

This component provides interfaces to hook up and service interrupts.

4.1 Resources Available

The PrOS kernel provides interrupt services based on irqs. This facility has been used for implementing this sub-component. Since PrOS does not provide facility of an argument to the callback function, only a single interrupt can be supported within the ISR component.

4.2 Dependencies

4.2.1 Subordinates

`MEM, TRC`

4.2.2 Preconditions

None.

4.3 Description

An `IsrObject` pointer is maintained to keep track of installed ISR.

A call to `ISR_Install ()` results in `ISR_Callback ()` to get registered as the ISR for the specified irq. When an interrupt occurs, `ISR_Callback ()` gets invoked and based on the irq number, it calls the user defined ISR.

4.4 Typedefs and Data Structures

4.4.1 IsrProc

Function prototype for an ISR. The user defined function to be invoked as an ISR must conform to this signature

Definition

```
typedef Void (*IsrProc ) (Pvoid refData) ;
```

Arguments

IN	Pvoid	refData

Data to be passed to ISR when invoked

Comments

This is the function signature for an interrupt service routine for DSP/BIOS Link.

4.4.2 IsrObject

Defines object to encapsulate the interrupt service routine. The definition is OS/platform specific.

Definition

```
struct IsrObject_tag {
    Uint32   signature ;
    Pvoid    refData   ;
    IsrProc  fnISR     ;
    INT      intId     ;
    Bool     installed ;
    Bool     enabled   ;
} ;
```

Fields

signature	Signature to identify this object. Is the literal string "ISR_".
refData	Argument to be passed to the Interrupt Service Routine.
fnISR	Actual Interrupt service routine.
irq	IRQ number for which ISR is to be installed.
installed	Flag to indicate the ISR is installed.
enabled	Flag to indicate the ISR is enabled

Comments

None.

Constraints

None.

See Also

None.

4.4.3 InterruptInfo

This structure encapsulates OS specific details of identifying an interrupt.

Definition

```
typedef struct InterruptObject_tag {
    Int32  intId ;
} InterruptObject ;
```

Fields

intId	Interrupt identifier.

Comments

On PrOS, an interrupt is identified through an IRQ number.

Constraints

None.

See Also

IsrObject

4.5 API Definition

4.5.1 ISR_Initialize

This function initializes the `IsrObject` pointer to NULL.

Syntax

```
DSP_STATUS ISR_Initialize () ;
```

Arguments

None.

Return Values

`DSP_SOK`	Successful initialization.
`DSP_EMEMORY`	Out of memory.

Comments

None.

Constraints

ISR sub-component must be initialized.

See Also

`ISR_Install`
`ISR_Finalize`

4.5.2 ISR_Finalize

This function finalizes all the resources used by this subcomponent and uninstalls ISR that have not yet been uninstalled.

Syntax

```
DSP_STATUS ISR_Finalize () ;
```

Arguments

None.

Return Values

`DSP_SOK`	Successful initialization.
`DSP_EMEMORY`	Out of memory.

Comments

None.

Constraints

ISR sub-component must be initialized.

See Also

ISR_Initialize

4.5.3 ISR_Create

This function creates an ISR object. It encapsulates the OS dependent definition of an ISR into the IsrObject and returns the caller.

Syntax

```
DSP_STATUS ISR_Create (IsrProc           fnISR,
                       Pvoid             refData,
                       InterruptObject * intInfo,
                       IsrObject **      isrObj) ;
```

Arguments

IN IsrProc FnISR

Interrupt service routine

IN Pvoid refData

Parameter to be passed to ISR

IN InterruptObject * intInfo

Interrupt information (OS and hardware dependent).

OUT IsrObject ** isrObj

Out argument for IsrObject

Return Values

DSP_SOK	Operation succesfully completed.
DSP_EMEMORY	Out of memory.
DSP_EINVALIDARG	Invalid arguments.

Comments

None.

Constraints

ISR sub-component must be initialized.

isrObj must be valid pointer.

intInfo must be a valid pointer.

fnISR must be a valid function pointer.

See Also

ISR_Delete

4.5.4 ISR_Delete

This function releases memory allocated for the isrObj.

Syntax

```
DSP_STATUS ISR_Delete (IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	isrObj

Isr object to be deleted

Return Value

DSP_SOK	Operation succesfully completed.
DSP_EPOINTER	Invalid isrObj pointer.
DSP_EMEMORY	Memory error.
DSP_EACCESSDENIED	Can't delete an IsrObject unless it is uninstalled.

Comments

None.

Constraints

ISR subcomponent must be initialized.

`isrObj` must be a valid object.

`isrObj` must not be installed.

See Also

`ISR_Create`

4.5.5 ISR_Install

Install an interrupt service routine defined by the `IsrObject` structure.

Syntax

```
DSP_STATUS ISR_Install (Void *       hostConfig,
                        IsrObject *  isrObj) ;
```

Arguments

IN	Void *	hostConfig

Host configuration to be used for installing the ISR

IN	IsrObject *	isrObj

ISR object to be installed.

Return Value

DSP_SOK	Operation succesfully completed.
DSP_EPOINTER	Invalid isrObj pointer.
DSP_EACCESSDENIED	ISR already installed for specified interruptInfo.

DSP_EFAIL	General error from GPP-OS.

Comments

This function makes a call to `request_irq ()` to install the interrupt specified by `isrObj`.

Constraints

ISR sub-component must be initialized.

`isrObj` must be valid.

See Also

`ISR_Func`
`ISR_Uninstall`

4.5.6 ISR_Uninstall

Uninstalls the interrupt service routine defined by `isrObj`.

Syntax

```
DSP_STATUS ISR_Uninstall (IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	isrObj

The interrupt object to be uninstalled

Return Value

DSP_SOK	Operation succesfully completed.
DSP_EPOINTER	Invalid isrObj pointer.
DSP_EACCESSDENIED	ISR is already uninstalled.
DSP_EFAIL	General error from GPP-OS.

Comments

None.

Constraints

ISR sub-component must be initialized.

`isrObj` must be a valid `IsrObject`.

See Also

`ISR_Install`

4.5.7 ISR_Disable

Disables an ISR associated with interrupt Id of `isrObject`.

Syntax

```
DSP_STATUS ISR_Disable (IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	isrObj

ISR Object indicating the isr to be disabled.

Return Value

DSP_SOK	Operation succesfully completed.
DSP_ENOTIMPL	Function not implemented.
DSP_EACCESSDENIED	ISR is not installed.
DSP_EFAIL	General error from GPP-OS.

Comments

This function calls dis_int function of the PrOS kernel to disable the specified interrupt.

Constraints

ISR sub-component must be initialized.

See Also

ISR_Enable
ISR_Install

4.5.8 ISR_Enable

Reactivates ISR based on the specified flags argument. The flags argument must be obtained with an earlier call to ISR_Disable ().

Syntax

```
DSP_STATUS ISR_Enable (IsrObject * isrObj) ;
```

Arguments

IN	IsrObject *	IsrObj

ISR Object indicating the isr to be enabled.

Return Value

DSP_SOK	Operation succesfully completed.
DSP_ENOTIMPL	Function not implemented.
DSP_EFAIL	General error from GPP-OS.

Comments

This function calls ena_int function of the PrOS kernel to enable the specified interrupt.

Constraints

`isrObj` must be a valid object.

See Also

`ISR_Disable`

4.5.9 ISR_GetState

Gets the status of ISR associated to this `isrObject`.

Syntax

```
DSP_STATUS ISR_GetState (IsrObject *  isrObj, ISR_State *  isrState) ;
```

Arguments

IN	`IsrObject *`	`isrObj`
	The ISR object	
OUT	`ISR_State *`	`isrState`
	Current status of the ISR	

Return Value

`DSP_SOK`	Operation succesfully completed.
`DSP_EPOINTER`	Invalid `isrObj argument`.
`DSP_EINVALIDARG`	Invalid `isrStatus` pointer.

Comments

None.

Constraints

ISR subcomponent must be initialized.

`isrObj` must be a valid `IsrObject`.

`isrState` must be a valid pointer.

See Also

`ISR_Install`
`ISR_Disable`
`ISR_Uninstall`
`ISR_Enable`

4.5.10 ISR_Callback

This function is registered as an interrupt handler for the irq on which user wants to register an ISR. The PrOS kernel calls this function when an interrupt occurs on the specified irq. This function, then, invokes the user specified interrupt service routine.

Syntax

```
void ISR_Callback (void) ;
```

Arguments

None.

Return Value

None.

Comments

None.

Constraints

This function is invoked by the PrOS kernel and is not invoked from anywhere else.

See Also

`ISR_Install`

5 KFILE

This component provides file system services to DSP/BIOS™ LINK similar to the ANSI C file system.

5.1 Resources Available

The PrOS kernel-PrFILE System provides function to perform file operations. The function pointers are the recommended method to perform file IO.

5.2 Dependencies

5.2.1 Subordinates

MEM, TRC

5.2.2 Preconditions

None.

5.3 Description

This component provides services to open, close, read from and write to a file. It also provides interface to reposition the file pointer.

5.4 Typedefs and Data Structures

5.4.1 KFileObject_tag

Definition

```
struct KFileObject_tag {
    PF_FILE *     fp        ;
    Uint32        signature ;
    Bool          isOpen    ;
    Uint32        size      ;
    Uint32        curPos    ;
} ;
```

Fields

`fp`	File pointer.
`signature`	Signature of the KFILE object
`isOpened`	Flag to track whether the file is opened
`size`	Size of this file
`curPos`	Current file position indicator

Comments

None.

Constraints

None.

See Also

None.

5.5 API Definition

5.5.1 KFILE_Initialize

Initializes the KFILE sub-component by allocating all resources.

Syntax

```
DSP_STATUS KFILE_Initialize () ;
```

Arguments

None.

Return Values

DSP_SOK	Successful initialization of component.
DSP_EMEMORY	Memory error, out of memory.
DSP_EFILE	File system error.

Comments

None.

Constraints

None.

See Also

KFILE_Finalize

5.5.2 KFILE_Finalize

Releases resources used by this sub-component.

Syntax

```
DSP_STATUS KFILE_Finalize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory.

Comments

None.

Constraints

Sub-component must be initialized.

See Also

KFILE_Initialize

5.5.3 KFILE_Open

Opens the specified file.

Syntax

```
DSP_STATUS KFILE_Open (CONST Char8 *        fileName,
                       CONST Char8 *        mode,
                       KFileObject **       fileHandle) ;
```

Arguments

IN	CONST Char8 *	fileName

Name of the file to be opened

IN	CONST Char8 *	mode

Mode for opening the file "read", "write", "append" etc

OUT	KFileObject **	fileHandle

Handle to the opened file if it could be opened successfully.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EFILE	File not found.
DSP_EINVALIDARG	Invalid arguments.

Comments

This function uses the pf_fopen () function provided by the PrFILE component (with PrOS) to open a file. The return value from this function is stored in the 'fp' field of the KFileObject structure. The 'fp' pointer is then used for other file IO operations. Constraints

Sub-component must be initialized.

fileName must be valid.

mode must be valid.

fileHandle must be valid.

Constraints

None.

See Also

KFILE_Close
KFILE_Read
KFILE_Write

5.5.4 KFILE_Close

Closes a file handle.

Syntax

```
DSP_STATUS KFILE_Close (KFileObject *   file) ;
```

Arguments

IN	KFileObject *	file

Handle of file to close, returned from KFILE_Open

Return Values

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid file object.
DSP_EFILE	File is not open.
DSP_EINVALIDARG	Invalid arguments.

Comments

This function uses pf_fclose () function to close the file.

Constraints

Sub-component must be initialized.

fileObj must be a valid handle to a file opened earlier.

See Also

KFILE_Open

5.5.5 KFILE_Read

Reads a specified number of items of specified size bytes from the file to a buffer.

Syntax

```
DSP_STATUS KFILE_Read (Char8 *        buffer
                       Uint32         size,
                       Uint32         count,
                       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_EPOINTER	Invalid file object.
DSP_EFILE	Error reading file.
DSP_EINVALIDARG	Invalid arguments.
DSP_ERANGE	The requested number of bytes is beyond EOF.

Comments

This function reads `size*count` bytes from the file and fills the buffer with the data read.

Constraints

`fileObj` must be a valid `KFileObject` pointer opened earlier.

Sub-component must be initialized.

See Also

`KFILE_Write`
`KFILE_Open`

5.5.6 KFILE_Seek

Repositions the file pointer according to the specified arguments.

Syntax

```
DSP_STATUS KFILE_Seek (KFileObject *   fileObj,
                       Int32           offset,
                       KFILE_FileSeek  origin) ;
```

Arguments

IN	KFileObject *	fileObj

Handle to file whose pointer is to be repositioned

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_EPOINTER	Invalid file object.
DSP_EFILE	File is not opened.
DSP_EINVALIDARG	Invalid arguments.
DSP_ERANGE	Offset and origin combination is beyond file size range.

Comments

None.

Constraints

`fileObj` must be a valid handle.

Subcomponent must be initialized.

See Also

`KFILE_Tell`

5.5.7 KFILE_Tell

Returns the current file pointer position for the specified file handle.

Syntax

```
DSP_STATUS KFILE_Tell (KFileObject * fileObj, 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_EPOINTER	Invalid file object.
DSP_EFILE	File is not opened.
DSP_EINVALIDARG	Invalid arguments.

Comments

None.

Constraints

Sub-component must be initialized.

`fileObj` must be a valid handle to a file opened earlier.

See Also

`KFILE_Seek`

6 MEM

This component provides dynamic memory allocation and deallocation services at run time.

6.1 Resources Available

The PrOS kernel provides `get_mpl ()` and `rel_mpl ()` to allocate and free memory.

`get_mpl()` and `rel_mpl()` functions ensure that the underlying physical memory is contiguous.

6.2 Dependencies

6.2.1 Subordinates

`None.`

6.2.2 Preconditions

None.

6.3 Description

DSP/BIOS Link uses PrOS Configured Memory POOL for its own memory requirements.

6.4 Typedefs and Data Structures

6.4.1 MemAllocAttrs

OS dependent attributes for allocating memory

Definition

```
typedef struct MemAllocAttrs_tag {
    Uint32 *    physicalAddress ;
} MemAllocAttrs ;
```

Fields

`physicalAddress`	Physical address of the allocated memory.

Comments

None.

Constraints

None.

See Also

`MemFreeAttrs`

6.4.2 MemFreeAttrs

OS dependent attributes for freeing memory

Definition

```
typedef struct MemFreeAttrs_tag {
    Uint32 *    physicalAddress ;
    Uint32      size ;
} MemFreeAttrs ;
```

Fields

`physicalAddress`	Physical address of the memory to be freed.
`size`	Size of the memory to be freed.

Comments

None.

Constraints

None.

See Also

`MemAllocAttrs`

6.4.3 MemMapInfo

OS dependent definition of the information required for mapping a memory region.

Definition

```
struct MemMapInfo_tag {
    Uint32   src  ;
    Uint32   size ;
    Uint32   dst  ;
} MemMapInfo ;
```

Fields

`src`	Address to be mapped.
`size`	Size of memory region to be mapped.
`dst`	Mapped address.

Comments

None.

Constraints

None.

See Also

`MemUnmapInfo`

6.4.4 MemUnmapInfo

OS dependent definition of the information required for unmapping a previously mapped memory region.

Definition

```
struct MemUnmapInfo_tag {
    Uint32  addr ;
    Uint32  size ;
} MemUnmapInfo ;
```

Fields

`addr`	Address to be unmapped. This is the address returned as `'dst'` address from a previous call to `MEM_Map ()` in the `MemMapInfo` structure.
`size`	Size of memory region to be unmapped.

Comments

None.

Constraints

None.

See Also

`MemMapInfo`

6.5 API Definition

6.5.1 MEM_Initialize

Initializes the MEM sub-component.

Syntax

```
DSP_STATUS MEM_Initialize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.

Comments

This function sets the initialized flag to TRUE.

Constraints

None.

See Also

None.

6.5.2 MEM_Finalize

Releases all resources used by this sub-component.

Syntax

```
DSP_STATUS MEM_Finalize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Memory error occurred.
DSP_EFAIL	General error from GPP-OS.

Comments

This function sets the initialized flag to FALSE.

Constraints

None.

See Also

None.

6.5.3 MEM_Alloc

Allocates the specified number of bytes.

Syntax

```
DSP_STATUS MEM_Alloc (Void ** ptr, Uint32 cBytes, 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

Not used for PrOS.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EINVAILDARG	Invalid argument.

Comments

This function uses `get_mpl ()`.

Constraints

MEM must be initialized.

`ptr` must be a valid pointer.

See Also

MEM_Free

6.5.4 MEM_Calloc

Allocates the specified number of bytes and clears them by filling it with zeroes.

Syntax

```
DSP_STATUS MEM_Calloc (Void ** ptr, Uint32 cBytes, Pvoid arg) ;
```

Arguments

OUT	Void **	ptr

Location where pointer to allocated memory is returned

IN	Uint32	cBytes

Number of bytes to allocate

IN OUT	Pvoid	arg

Not used for PrOS.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EINVAILDARG	Invalid argument.

Comments

None.

Constraints

MEM must be initialized.

ptr must be a valid pointer.

See Also

None.

6.5.5 MEM_Free

Frees up the allocated chunk of memory. (Not Implemented for PrOS.)

Syntax

```
DSP_STATUS MEM_Free (Pvoid * ptr, Pvoid arg) ;
```

Arguments

IN	Pvoid *	ptr

Pointer to pointer to start of memory to be freed

IN	Pvoid	arg

Not used for PrOS.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.
DSP_EINVAILDARG	Invalid argument.

Comments

None.

Constraints

MEM must be initialized.

ptr must be a valid pointer.

See Also

None.

6.5.6 MEM_Map

Maps a specified memory area into the GPP virtual space. (Not Implemented for PrOS.)

Syntax

```
DSP_STATUS MEM_Map (MemMapInfo * mapInfo) ;
```

Arguments

IN OUT	MemMapInfo *	mapInfo

Data required for creating the mapping

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Could not map the given memory address.

Comments

None.

Constraints

mapInfo pointer must be valid.

See Also

MEM_Unmap

6.5.7 MEM_Unmap

Unmaps the specified memory area. (Not Implemented for PrOS.)

Syntax

```
DSP_STATUS MEM_Unmap (MemUnmapInfo * unmapInfo) ;
```

Arguments

IN	MemUnmapInfo *	unmapInfo

Information required for unmapping a memory area

Return Values

DSP_SOK	Operation successfully completed.

Comments

None.

Constraints

`unmapInfo` pointer must be valid.

See Also

`MEM_Map`

6.5.8 MEM_Copy

Copies data between the specified memory areas

Syntax

```
DSP_STATUS
MEM_Copy (Uint8 * dst, Uint8 * src, Uint32 len, Endianism endian) ;
```

Arguments

IN	`Uint8 *`	`dst`

Destination address

IN	`Uint8 *`	`src`

Source address

IN	`Uint32`	`len`

Length of data to be coiped.

IN	`Endianism`	`endian`

Endianism

Return Values

`DSP_SOK`	Operation successfully completed.

Comments

None.

Constraints

`dst` and `src` must be valid pointers.

See Also

`None.`

6.5.9 MEM_Debug

Prints debug information for MEM.

Syntax

```
Void MEM_Debug () ;
```

Arguments

None.

Return Values

None.

Comments

None.

Constraints

This function can only be used in debug builds**.**

See Also

None.

7 PRCS

7.1 Resources Available

PrOS has single process space, users can create multiple tasks. The implementation shall assume that the process context are same for all tasks and the task will be differentiated on the basis of their id's.

7.2 Dependencies

7.2.1 Subordinates

TRC.

7.2.2 Preconditions

None.

7.3 Description

None.

7.4 Typedefs and Data Structures

7.4.1 PrcsObject

Structure to store information regarding current process/thread.

Definition

```
struct PrcsObject_tag {
    Uint32 signature          ;
    Void * handleToProcess    ;
    Void * handleToThread     ;
    Int32  priorityOfProcess  ;
    Int32  priorityOfThread   ;
};
```

Fields

signature	Signature of this structure.
handleToProcess	Handle to current process.
handleToThread	Handle to current thread.
priorityOfProcess	Priority of current process.
priorityOfThread	Priority of current thread.

Comments

None.

7.5 API Definition

7.5.1 PRCS_Initialize

Initializes the PRCS sub-component.

Syntax

```
DSP_STATUS PRCS_Initialize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EMEMORY	Out of memory error.

Comments

None.

Constraints

None.

See Also

None.

7.5.2 PRCS_Finalize

Releases resources used by the PRCS sub-component.

Syntax

```
DSP_STATUS PRCS_Finalize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.

Comments

None.

Constraints

None.

See Also

None.

7.5.3 PRCS_Create

Creates a `PrcsObject` and populates it with information to identify the client.

Syntax

```
DSP_STATUS PRCS_Create (PrcsObject ** prcsObj);
```

Arguments

OUT	PrcsObject **	prcsObj

OUT argument to store the created object

Return Value

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument.

Comments

None.

Constraints

`prcsObj` must be a valid pointer.

See Also

`PRCS_Delete`

7.5.4 PRCS_Delete

Frees up resources used by the specified object.

Syntax

```
DSP_STATUS PRCS_Delete(PrcsObject * prcsObj)
```

Arguments

OUT	PrcsObject *	prcsObj

Object to be deleted.

Return Value

DSP_SOK	Operation successfully completed.
DSP_EPOINTER	Invalid prcsObj.

Comments

None.

Constraints

`prcsObj` must be a valid object.

See Also

PRCS_Create

7.5.5 PRCS_IsEqual

Compares two clients to check if they are "equal". Equality is defined by implementation on the specific OS port.

Syntax

```
DSP_STATUS PRCS_IsEqual (PrcsObject * client1,
                         PrcsObject * client2,
                         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.

Comments

None.

Constraints

client1 must be a valid object.

client2 must be a valid object.

isEqual must be a valid pointer.

See Also

PRCS_Create

7.5.6 PRCS_IsSameContext

Checks if the two clients share same context.

Syntax

```
DSP_STATUS  PRCS_IsSameContext (PrcsObject *  client1,
                                PrcsObject *  client2,
                                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.

Comments

Since PrOS has only one process space, so it returns true always.

Constraints

client1 must be a valid object.

client2 must be a valid object.

isSame must be a valid pointer.

See Also

PRCS_Create

8 PRINT

This subcomponent provides printing services to DSP/BIOS™ LINK.

8.1 Resources Available

PrOS PSP provides `kprintf ()` for printing messages on the target terminal. This sub-component uses these functions to provide the required services.

8.2 Dependencies

8.2.1 Subordinates

`None.`

8.2.2 Preconditions

None.

8.3 Description

`kprintf ()` is implemented as a wrapper over the UART functions.

8.4 API Definition

8.4.1 PRINT_Initialize

Initializes the PRINT sub-component.

Syntax

```
DSP_STATUS PRINT_Initialize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS

Comments

None.

Constraints

None.

See Also

None.

8.4.2 PRINT_Finalize

Releases resources used by this sub-component.

Syntax

```
DSP_STATUS PRINT_Finalize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS

Comments

None.

Constraints

None.

See Also

None.

8.4.3 PRINT_Printf

Provides standard printf functionality abstraction.

Syntax

```
Void PRINT_Printf (Pstr format, ...) ;
```

Arguments

`IN` `Pstr` `format`

Format string to be used for formatted display

`IN` `...`

Variable list of arguments

Return Values

None.

Comments

None.

Constraints

None.

See Also

None.

9 SYNC

This component provides synchronization APIs to DSP/BIOS™ LINK. Synchronization APIs of this component can be classified broadly in two categories.

1. Semaphores
2. Event Based Synchronization
3. Critical Section and spin-lock based Mutual Exclusion

9.1 Resources Available

PrOS provide APIs for counting and binary semaphores. These have been used in implementing the functionality of this sub-component.

In addition, the PrOS facility for semaphore has been used to provide the highest level of protection against tasks, DPCs and ISRs.

9.2 Dependencies

9.2.1 Subordinates

`TRC, MEM`.

9.2.2 Preconditions

None.

9.3 Description

None.

9.4 Constants & Enumerations

9.4.1 SyncSemType

This enumeration defines the possible types of semaphores that can be created.

Definition

```
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.

Comments

The semaphore type is stored within the `SyncSemObject`. It indicates the type of the semaphore to be created, when passed to `SYNC_CreateSEM ()` through the flags field of the `SyncAttrs`.

Constraints

None.

See Also

SyncAttrs
SyncSemObject
SYNC_CreateSEM ()

9.5 Typedefs and Data Structures

9.5.1 SyncAttrs

This object contains various attributes of SYNC object.

Definition

```
typedef struct SyncAttrs_tag {
    Uint16   flag     ;
} SyncAttrs ;
```

Fields

flags	This flag is used by the various SYNC functions and its usage is dependent on the function using it.

Comments

None.

See Also

SYNC_OpenEvent ()
SYNC_CreateSEM ()

9.5.2 SyncEvObject

This object is used for various event related API.

Definition

```
struct SyncEvObject_tag {
    Uint32            signature       ;
    struct semaphore  eventSem        ;
    Bool              timeoutOccurred ;
} ;
```

Fields

signature	Used for identification of this object.
eventSem	OS specific semaphore object.
timeoutOccurred	Indicates that timeout had occurred.

Comments

None.

See Also

None.

9.5.3 SyncCSObject

This object is used by various CS API's.

Definition

```
struct SyncCsObject_tag {
    Uint32            signature ;
```

```
    ID                mtxId     ;
} ;
```

Fields

Signature	Used for identification of this object.
mtxId	ID of mutex.

Comments

None.

See Also

None.

9.5.4 SyncSemObject

This object is used by various SEM API's.

Definition

```
struct SyncSemObject_tag {
    Uint32            signature       ;
    SyncSemType       semType         ;
    Bool              isSemAvailable  ;
    Bool              timeoutOccurred ;
} ;
```

Fields

signature	For identification of this object.
semType	Indicates the type of the semaphore (binary or counting).
isSemAvailable	Flag to indicate if the binary semaphore is available. If flag is TRUE then semaphore is available. If flag is FALSE then semaphore is not available.
timeoutOccurred	Indicates that timeout had occurred.

Comments

None.

See Also

None.

9.6 API Definition

9.6.1 SYNC_Initialize

Initializes SYNC sub-component by allocating all resources.

Syntax

```
DSP_STATUS SYNC_Initialize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.

Comments

None.

Constraints

None.

See Also

SYNC_Finalize

9.6.2 SYNC_Finalize

Releases all the resources used by the SYNC sub-component.

Syntax

```
DSP_STATUS SYNC_Finalize () ;
```

Arguments

None.

Return Values

DSP_SOK	Operation successfully completed.

Comments

None.

Constraints

None.

See Also

SYNC_Initialize

9.6.3 SYNC_OpenEvent

Creates and initializes an event object for thread synchronization. The event is initialized to a non-signaled state.

Syntax

```
DSP_STATUS SYNC_OpenEvent (SyncEvObject ** event, 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.
DSP_EFAIL	General error from GPP-OS.
DSP_EMEMORY	Operation failed due to insufficient memory.
DSP_EPOINTER	Invalid pointer passed

Comments

None.

Constraints

`event` must be valid.

`attr` must be valid.

See Also

`SYNC_CloseEvent`

9.6.4 SYNC_CloseEvent

Closes the handle corresponding to an event. It also frees the resources allocated, if any, during call to `SYNC_OpenEvent ()`.

Syntax

```
DSP_STATUS SYNC_CloseEvent (SyncEvObject * event) ;
```

Arguments

IN	SyncEvObject *	event

Event to be closed

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed

Comments

None.

Constraints

`event` must be a valid object.

See Also

`SYNC_OpenEvent`

9.6.5 SYNC_ResetEvent

Resets the synchronization object to non-signaled state.

Syntax

```
DSP_STATUS SYNC_ResetEvent (SyncEvObject * event) ;
```

Arguments

IN	SyncEvObject *	event

Event to be reset

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed

Comments

None.

Constraints

`event` must be a valid object.

See Also

`SYNC_SetEvent`

9.6.6 SYNC_SetEvent

Sets the state of synchronization object to signaled and unblocks all threads waiting for it.

Syntax

```
DSP_STATUS SYNC_SetEvent (SyncEvObject * event);
```

Arguments

IN	SyncEvObject *	event

Event to be set

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed

Comments

None.

Constraints

event must be a valid object.

See Also

SYNC_ResetEvent

9.6.7 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

```
DSP_STATUS SYNC_WaitOnEvent (SyncEvObject * event, Uint32 timeout) ;
```

Arguments

IN	SyncEvObject *	event
	Event to be waited upon	
IN	Uint32	timeout
	Timeout value	

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.
DSP_ETIMEOUT	Timeout occured while performing operation.

Comments

This function ‘block’s and does not 'spin' while waiting on the event.

Constraints

event must be a valid object.

See Also

SYNC_WaitOnMultipleEvents

9.6.8 SYNC_WaitOnMultipleEvents

Waits on multiple events. Returns when any of the events is set.

Syntax

```
DSP_STATUS SYNC_WaitOnMultipleEvents (SyncEvObject ** syncEvents,
                                      Uint32          count,
                                      Uint32          timeout,
                                      Uint32 *        index) ;
```

Arguments

IN	SyncEvObject **	syncEvents

Array of events to be waited upon

IN	Uint32	count

Number of events

IN	Uint32	timeout

Timeout value for wait

OUT	Uint32 *	index

OUT argument to store the index of event that is set

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.
DSP_ETIMEOUT	Timeout occured while performing operation.

Comments

This function is not implemented in the PrOS port of OSAL.

Constraints

event must be a valid object.

See Also

SYNC_WaitOnEvent

9.6.9 SYNC_CreateCS

Initializes the Critical section structure.

Syntax

```
DSP_STATUS SYNC_CreateCS (SyncCsObject ** cSObj) ;
```

Arguments

OUT	SyncCsObject **	cSObj

Structure to be initialized.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EMEMORY	Operation failed due to insufficient memory.
DSP_EPOINTER	Invalid pointer passed.

Comments

This function creates a semaphore object to implement critical sections APIs.

Constraints

cSObj must be a valid object.

See Also

SYNC_DeleteCS

9.6.10 SYNC_DeleteCS

Deletes the critical section object.

Syntax

```
DSP_STATUS SYNC_DeleteCS (SyncCsObject * cSObj) ;
```

Arguments

IN SyncCsObject * cSObj

Critical section to be deleted.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.

Comments

None.

Constraints

cSObj must be a valid object.

See Also

SYNC_CreateCS

9.6.11 SYNC_EnterCS

This function enters the critical section that is passed as argument to it. After successful return of this function no other thread can enter until this thread exits the critical section by calling SYNC_LeaveCS ().

Syntax

```
DSP_STATUS SYNC_EnterCS (SyncCsObject * cSObj) ;
```

Arguments

IN SyncCsObject * cSObj

Critical section to enter.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EPOINTER	Invalid pointer passed.

Comments

This function does a 'loc_mtx ()' call on the mutex.

Constraints

cSObj must be a valid object.

See Also

SYNC_LeaveCS

9.6.12 SYNC_LeaveCS

This function makes the critical section available for other threads to enter.

Syntax

```
DSP_STATUS SYNC_LeaveCS (SyncCsObject * cSObj) ;
```

Arguments

IN SyncCsObject * CSObj

Critical section to leave.

Return Values

DSP_SOK	Operation successfully completed.
DSP_EFAIL	General error from GPP-OS.
DSP_EMEMORY	Operation failed due to insufficient memory
DSP_EPOINTER	Invalid pointer passed.

Comments

This function does an 'uloc_mtx ()' call on the semaphore to allow access to the critical section by other waiting threads.

Constraints

cSObj must be a valid object.

See Also

SYNC_EnterCS

9.6.13 SYNC_CreateSEM

Creates the semaphore object.

Syntax

```
DSP_STATUS SYNC_CreateSEM (SyncSemObject ** semObj, 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.

Comments

None.

Constraints

semObj must be a valid object.

attr must not be NULL.

See Also

SYNC_DeleteSEM

9.6.14 SYNC_DeleteSEM

Deletes the semaphore object.

Syntax

```
DSP_STATUS SYNC_DeleteSEM (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.

Comments

None.

Constraints

`semObj` must be a valid object.

See Also

`SYNC_CreateSEM`

9.6.15 SYNC_WaitSEM

This function waits on the semaphore.

Syntax

```
DSP_STATUS SYNC_WaitSEM (SyncSemObject * semObj, 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.

Comments

None.

Constraints

`semObj` must be a valid object.

See Also

`SYNC_SignalSEM`

9.6.16 SYNC_SignalSEM

This function signals the semaphore and makes it available for other threads.

Syntax

```
DSP_STATUS SYNC_SignalSEM (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.

Comments

None.

Constraints

`semObj` must be a valid object.

See Also

`SYNC_WaitSEM`

9.6.17 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.

Syntax

```
Uint32 SYNC_SpinLockStart () ;
```

Arguments

None.

Return Values

`irqFlags` On success.

Comments

This API can be called from DPC context. loc_cpu is used for implementing this API.

Constraints

None.

See Also

`SYNC_SpinLockEnd`

9.6.18 SYNC_SpinLockEnd

End protection of code through spin lock with all ISRs disabled.

Syntax

```
Void SYNC_SpinLockEnd (Uint32 irqFlags) ;
```

Arguments

`IN` `Uint32` `irqFlags`

Pointer to semaphore object to be signalled.

Return Values

None.

Comments

This API can be called from DPC context. uloc_cpu is used for implementing this API.

Constraints

None.

See Also

`SYNC_SpinLockStart`

9.6.19 SYNC_ProtectionStart

Marks the start of protected code execution.

Syntax

```
Void SYNC_ProtectionStart () ;
```

Arguments

None.

Return Values

None.

Comments

DSP/BIOS Link implements DPC using `task`. This function achieves protection by disabling DPCs.

Constraints

None.

See Also

`None.`

9.6.20 SYNC_ProtectionEnd

Marks the end of protected code execution.

Syntax

```
Void SYNC_ProtectionEnd () ;
```

Arguments

None.

Return Values

None.

Comments

DSP/BIOS Link implements DPC using tasks. This function enables DPCs.

Constraints

None.

See Also

`None.`

10 TRC

This subcomponent provides the functionality to print debug messages on the target terminal.

10.1 Resources Available

This subcomponent doesn't need any operating system specific resources. It uses services from the PRINT subcomponent to print debug messages on the target terminal.

10.2 Dependencies

10.2.1 Subordinates

`MEM, PRINT.`

10.2.2 Preconditions

None.

10.3 Description

The TRC Object is a global structure, common for all components and their corresponding sub-components. When debug messages need to be printed, the TRC object is first checked to see if each component and its subcomponents are permitted to print. For each enabled component and subcomponent the corresponding debug messages are printed depending on the severity associated with the individual messages and the requested severity that is set in the TRC Object.

10.4 Typedefs and Data Structures

10.4.1 TrcObject

TRC Object that stores the severity and component and subcomponent maps on a global level.

Definition

```
typedef struct TrcObject_tag {
    Uint16        components                    ;
    Uint16        level                         ;
    Uint16        subcomponents[MAX_COMPONENTS] ;
} TrcObject ;
```

Fields

components	Indicates which components (PMGR, LDRV, OSAL) are enabled to print debug messages**.**
level	Defines the level of severity used to decide the level of debug printing.
subcomponents	Indicates which subcomponents (PROC, CHNL, IO, DSP in the case of LDRV) are enabled to print debug messages**.**

Comments

This object stores information related to a debug trace mechanism.

MAX_COMPONENTS indicates the maximum number of components.

10.5 API Definition

10.5.1 TRC_0Print

Prints a null terminated character string based on its severity, the subcomponent and component it is associated with.

Syntax

```
Void TRC_0Print (Uint32   componentMap,
                 Uint16   severity,
                 Char8 *  debugString) ;
```

Arguments

IN Uint32 componentMap

The component and 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.

Comments

This function is used to print only a string without any additional arguments.

Constraints

The character string is valid.

See Also

TRC_1Print
TRC_2Print
TRC_3Print
TRC_4Print
TRC_5Print
TRC_6Print

10.5.2 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

```
Void TRC_1Print (Uint32   componentMap,
                 Uint16   severity,
                 Char8 *  debugString,
                 Uint32   argument1) ;
```

Arguments

IN	Uint32	componentMap

The component and 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.

Comments

This function is used to print a string with one integer argument.

Constraints

The character string is valid.

See Also

TRC_0Print
TRC_2Print
TRC_3Print
TRC_4Print
TRC_5Print
TRC_6Print

10.5.3 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

```
Void TRC_2Print (Uint32   componentMap,
                 Uint16   severity,
                 Char8 *  debugString,
                 Uint32   argument1,
                 Uint32   argument2) ;
```

Arguments

IN	Uint32	componentMap

The component and 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.

Comments

This function is used to print a string with two integer arguments.

Constraints

The character string is valid.

See Also

```
TRC_0Print
TRC_1Print
TRC_3Print
TRC_4Print
TRC_5Print
TRC_6Print
```

10.5.4 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

```
Void TRC_3Print (Uint32   componentMap,
                 Uint16   severity,
                 Char8 *  debugString,
                 Uint32   argument1,
                 Uint32   argument2,
                 Uint32   argument3) ;
```

Arguments

```
IN      Uint32                  componentMap
```

The component and 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.

Comments

This function is used to print a string with three integer arguments.

Constraints

The character string is valid.

See Also

```
TRC_0Print
TRC_1Print
TRC_2Print
TRC_4Print
TRC_5Print
TRC_6Print
```

10.5.5 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

```
Void TRC_4Print (Uint32   componentMap,
                 Uint16   severity,
                 Char8 *  debugString,
                 Uint32   argument1,
                 Uint32   argument2,
                 Uint32   argument3,
                 Uint32   argument4) ;
```

Arguments

```
IN      Uint32                  componentMap
```

The component and 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.

Comments

This function is used to print a string with four integer arguments.

Constraints

The character string is valid.

See Also

TRC_0Print
TRC_1Print
TRC_2Print
TRC_3Print
TRC_5Print
TRC_6Print

10.5.6 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

```
Void TRC_5Print (Uint32   componentMap,
                 Uint16   severity,
                 Char8 *  debugString,
                 Uint32   argument1,
                 Uint32   argument2,
                 Uint32   argument3,
                 Uint32   argument4,
                 Uint32   argument5) ;
```

Arguments

IN Uint32 componentMap

The component and 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.

Comments

This function is used to print a string with five integer arguments.

Constraints

The character string is valid.

See Also

TRC_0Print
TRC_1Print
TRC_2Print
TRC_3Print
TRC_4Print
TRC_6Print

10.5.7 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

```
Void TRC_6Print (Uint32   componentMap,
                 Uint16   severity,
                 Char8 *  debugString,
                 Uint32   argument1,
                 Uint32   argument2,
```

```
                    Uint32   argument3,
                    Uint32   argument4,
                    Uint32   argument5,
          Uint32   argument6) ;
```

Arguments

IN	Uint32	componentMap	
	The component and 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.

Comments

This function is used to print a string with six integer arguments.

Constraints

The character string is valid.

See Also

TRC_0Print
TRC_1Print
TRC_2Print
TRC_3Print

TRC_4Print
TRC_5Print

10.5.8 TRC_Enable

Enables debug prints on a component and sub-component level.

Syntax

```
DSP_STATUS TRC_Enable (Uint32 componentMap) ;
```

Arguments

IN	Uint32	componentMap

The component and subcomponent map

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument to function call.
DSP_EFAIL	Operation not successful.

Comments

Note that this function must be used to enable subcomponents belonging to the same component. On the same lines each component must be enabled individually.

Constraints

None.

See Also

TRC_Disable
TRC_SetSeverity

10.5.9 TRC_Disable

Disables debug prints on a component and sub-component level.

Syntax

```
DSP_STATUS TRC_Disable (Uint32 componentMap) ;
```

Arguments

IN	Uint32	componentMap

The component and subcomponent map

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument to function call.
DSP_EFAIL	Operation not successful.

Comments

Note that this function must be used to disable subcomponents belonging to the same component. On the same lines each component must be disabled individually.

Constraints

None.

See Also

TRC_Enable
TRC_SetSeverity

10.5.10 TRC_SetSeverity

Sets the severity level of the required debug prints.

Syntax

```
DSP_STATUS TRC_SetSeverity (Uint16 level) ;
```

Arguments

IN	Uint32	level

The severity level of the debug prints required

Return Values

DSP_SOK	Operation successfully completed.
DSP_EINVALIDARG	Invalid argument to function call.
DSP_EFAIL	Operation not successful.

Comments

None.

Constraints

None.

See Also

TRC_Enable
TRC_Disable