DSP/BIOS™ LINK

DSP Executable Loader

LNK 040 DES

Version 1.00

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 overall design and architecture of the Loader used to parse and load DSP binaries for DSP/BIOS™ LINK.

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

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

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

- q This document is still 'Work In Progress' and the contents may change frequently.

1.2 Terms & Abbreviations

DSPLINK	DSP/BIOS™ LINK
PMGR	Processor Manager
O	This bullet indicates important information. Please read such text carefully.
q	This bullet indicates additional information.

O This is important information.

q This is additional information.

1.3 References

1.	LNK 002 ARC	DSP/BIOS Link High Level Architecture Version 1.02 dated JUL 15, 2003
2.	LNK 010 DES	DSP/BIOS Link Processor Manager Version 1.11 dated OCT 08, 2002

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.

DSP Executable Loader provides the file Loading services to the DSP/BIOS™ LINK. DSPLINK is designed to support heterogeneous DSP's and therefore this component supports multiple loaders.

2 Requirements

The basic requirements for the loader component can be summarized as below:

R8	DSP/BIOS Link shall provide the means for the GPP to load a fully linked and located base DSP executable program into DSP memory and start it running.
R9	DSP/BIOS Link must provide an option to automatically load a DSP base image onto the DSP upon GPP/DSP device boot-up.
R10	The DSP loading capability shall support the OEM to perform field upgrades of new DSP base images on a deployed device.
R11	The GPP must be able to pass DSP/BIOS main () program arguments and global environment variables to the DSP at the time of loading.
R12	The DSP executable format must allow efficient loading, and allow all symbols, except those designated as necessary to support DSP/BIOS Link operation, to be stripped when deployed.
R13	DSP/BIOS Link shall allow the DSP executable to load into a combination of the DSP's internal and external memory.
R14	DSP/BIOS Link GPP APIs shall allow the same, or different, DSP executable images to be loaded onto specific DSPs connected via the Link.
R16	Identification: To support customization of multiple copies of the same software running on separate DSPs, there must be a means for the GPP to communicate the processor ID to each DSP.

3 Assumptions

- Only COFF file format loader is supported, though users can plug-in their own loaders for different file formats.
- The DSP application shall reserve sufficient memory for the '.args' section. This is required to allow the GPP to specify arguments to the 'main ()' function on the DSP.
- In the current phase only the COFF loader shall be supported.

4 Constraints

The design of the Loader component in DSPLINK is constrained by the following:

- The Loader component must comply with the interfaces expected by the PMGR component in DSPLINK.

5 High Level Design

5.1 Component interaction

The Processor Manager component uses services from a Loader for boot loading a DSP. The functions required for boot loading are exposed to PMGR through a function pointer table, which is a configurable attribute and is specified through the link driver configuration on a per DSP basis.

The component interaction diagram gives an overview of the interaction of the Loader component with other components of DSPLINK.

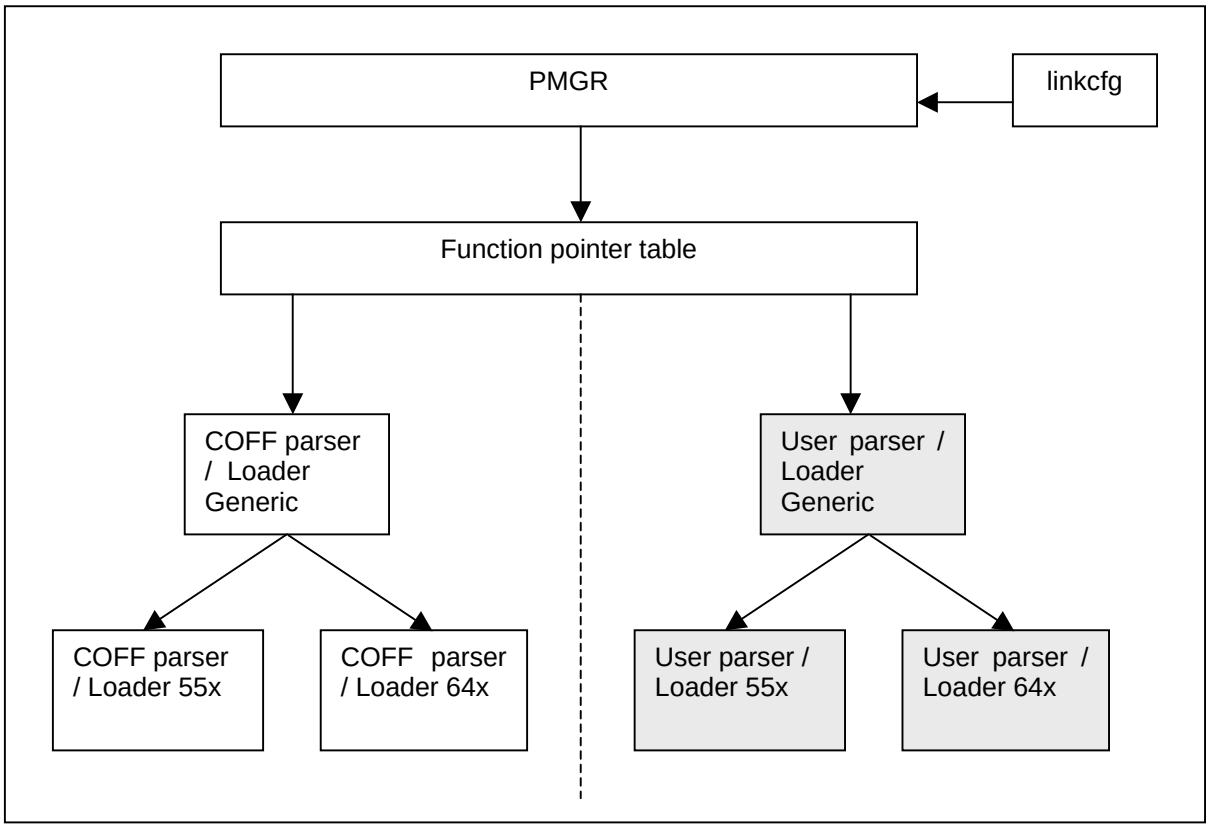

Figure 1. GPP-side component interaction diagram

This approach provides good level of plugability, since this allows PMGR component to be agnostic to the type of loader being used. Loaders for different file formats as well as dynamic loaders are easily pluggable once they expose a set of functions through function pointer table in CFG for a DSP.

The following configurable attributes in CFG are specified to plug a loader into DSPLINK:

1. `LOADERNAME`: This field is for debugging purpose only.
2. `ARCHITECTURE`: This field specifies the architecture of the target DSP. The loader uses this field to select parsing functions for the architecture. The loader also uses this field to check the validity of the DSP executable with respect to its architecture.

3. LOADER: This field specifies the address of the function pointer table used for loading the DSP executable onto the DSP.

5.1.1 Generic COFF Loader

The Generic COFF Loader implements the file loading functions common to all the supported architectures (presently 55x and 64x). It implements the functionality to load a complete coff file onto the DSP or load a specific section in the coff file. It also provides a function to print debug information.

The COFF Loader consists of generic and architecture specific functions. The architecture specific function provides services to read architecture specific details from a DSP executable. The generic functions utilize these services to interpret and load the DSP executable onto DSP.

The COFF loader provides interfaces to load a complete DSP executable to the DSP or load a specific section onto the DSP.

1. For loading the complete DSP executable onto the DSP, the loader parses the specified file and loads it section by section. A COFF section header specifies whether the section is loadable. If the section is loadable, it is loaded. It stores the start address of the DSP executable and passes it back to PMGR.
2. For loading a specific section the loader ensures that the section is loadable before loading it. If the section is not loadable an error is returned to PMGR.

The loader allows arguments to be specified to the `main ()` function of the DSP executable. The COFF format contains a named section, `'.args'`, to specify user arguments for the `main ()` function. The loader writes these arguments to the load address of this section.

6 Constants & Enumerations

6.1 ProcArchitecture

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

Definition

```
typedef enum {
    ProcArchitecture_Unknown =  0,
    ProcArchitecture_C55x     = 1,
    ProcArchitecture_C64x     = 2
} ProcArchitecture ;
```

Fields

ProcArchitecture_Unknown	Flag to indicate that the architecture is not supported.
ProcArchitecture_C55x	Flag to indicate that the architecture is C55x.
ProcArchitecture_C64x	Flag to indicate that the architecture is C64.

Comments

None.

Constraints

None.

See Also

None.

6.2 SWAP_LOCATION

It defines the location in COFF file where swap information is kept.

Definition

```
#define CONST1    100
```

Comments

None.

Constraints

None.

See Also

None.

6.3 SECT_DSECT

It defines the identifier for dummy section.

Definition

```
#define SECT_DSECT  0x0001
```

Comments

None.

Constraints

None.

See Also

None.

6.4 SECT_NOLOAD

It defines the identifier for a no load section.

Definition

```
#define  SECT_NOLOAD 0x0002
```

Comments

None.

Constraints

None.

See Also

None.

6.5 SECT_BSS

It defines the identifier for a BSS section.

Definition

```
#define  SECT_BSS 0x0080
```

Comments

None.

Constraints

None.

See Also

None.

6.6 SECT_COPY

It defines the identifier for a COPY section.

Definition

```
#define SECT_COPY 0x0010
```

Comments

None.

Constraints

None.

See Also

None.

6.7 SIZE_OPT_HDR_LOC

It defines the location in file header for number of bytes in optional header.

Definition

```
#define     SIZE_OPT_HDR_LOC     16
```

Comments

None.

Constraints

None.

See Also

None.

6.8 COFF_NAME_LEN

It defines the length of name.

Definition

```
#define  COFF_NAME_LEN  8
```

Comments

None.

Constraints

None.

See Also

None.

6.9 SYMTAB_OFFSET

It defines the offset in file header where symbol table details are present.

Definition

```
#define     SYMTAB_OFFSET   8
```

Comments

None.

Constraints

None.

See Also

None.

6.10 NUM_SECT_OFFSET

It defines the offset in file header where number of sections is present.

Definition

```
#define     NUM_SECT_OFFSET   2
```

Comments

None.

Constraints

None.

See Also

None.

6.11 COFF_MAGIC_64x

It defines the magic number to identify 64x COFF file format.

Definition

```
#define   COFF_MAGIC_64x    0x0099
```

Comments

None.

Constraints

None.

See Also

COFF_MAGIC_55x

6.12 COFF_MAGIC_55x

It defines the magic number to identify 55x COFF file format.

Definition

```
#define  COFF_MAGIC_55x  0x009c
```

Comments

None.

Constraints

None.

See Also

COFF_MAGIC_64x

7 Typedefs & Data Structures

7.1 CoffFileHeader

File header for a COFF file.

Definition

```
typedef struct CoffFileHeader_tag {
    Uint16  version          ;
    Uint16  numSections      ;
    Int32   dateTime         ;
    Int32   fpSymTab         ;
    Int32   numSymTabEntries ;
    Uint16  numBytesOptHeader ;
    Uint16  flags            ;
    Uint16  targetId         ;
} CoffFileHeader ;
```

Fields

version	Version ID. Indicates the version of the COFF file structure
numSections	Number of section headers
dateTime	Time and date stamp. Indicates when the file was created
fpSymTab	Symbol table's starting location in file
numSymTabEntries	Number of entries in the symbol table
NumBytesOptHeader	Number of bytes in the optional header. This field is either 0 or 28. If it is 0, there is no optional file header
flags	Flags (see the File Header Flags table).
targetId	Target ID. Magic number indicates the file can be executed in a particular system. This field is checked for validating the support of supplied file

Comments

None.

Constraints

None.

See Also

None.

7.2 CoffContext

This is the structure defining the context of parser. This object is created on initialization of this sub component and it is required to be passed as a parameter for any subsequent function call.

Definition

```
typedef struct CoffContext_tag {
    KFileObject * fileObj    ;
    Uint32        startAddr  ;
    Bool          isSwapped  ;
} CoffContext ;
```

Fields

`fileObj`	File object for the DSP base image file.
`startAddr`	Entry point address for the DSP base image file.
`isSwapped`	Flag to indicate if the file data is swapped.

Comments

None.

Constraints

None.

See Also

None.

7.3 CoffOptHeader

This is the structure defining the optional header for coff file format.

Definition

```
typedef struct CoffOptHeader_tag {
    Int16 magic          ;
    Int16 version        ;
    Int32 sizeExeCode    ;
    Int32 sizeInitData   ;
    Int32 sizeUninitData ;
    Int32 entry          ;
    Int32 addrExe        ;
    Int32 addrInitData   ;
} CoffOptHeader ;
```

Fields

`Magic`	Optional file header magic number
`version`	Version stamp.
`sizeExeCode`	Size (in bytes) of executable code.
`sizeInitData`	Size (in bytes) of initialized data.
`sizeUninitData`	Size (in bytes) of uninitialized data.
`Entry`	Entry point.
`addrExe`	Beginning address of executable code.

addrInitData	Beginning address of initialized data

Comments

None.

Constraints

None.

See Also

None.

7.4 CoffSectionHeader

This is the structure defining the section header for COFF file format.

Definition

```
typedef struct CoffSectionHeader_tag {
    Char8   name [COFF_NAME_LEN] ;
    Int32   physicalAddress      ;
    Int32   virtualAddress       ;
    Int32   size                 ;
    Int32   fpRawData            ;
    Int32   fpReloc              ;
    Int32   fpLineNum            ;
    Uint32  numReloc             ;
    Uint32  numLine              ;
    Uint32  flags                ;
    Uint16  reserved             ;
    Uint16  memPageNum           ;
    Bool    isLoadSection        ;
    Char8 * data                 ;
} CoffSectionHeader ;
```

Fields

Name	This field contains one of the following: 1) An 8-character section name, padded with nulls, or 2) A pointer into the string table if the section name is longer than 8 characters. In the latter case the first four bytes of the field are 0.
physicalAddress	Section's physical address.
virtualAddress	Section's virtual address.
Size	Section's size in bytes.
fpRawData	File pointer to raw data.
fpReloc	File pointer to relocation entries.
fpLineNum	File pointer to line-number entries.

numReloc	Number of relocation entries.
numLine	Number of line-number entries.
Flags	Flags (see the Section Header Flags table)
reserved	Reserved.
MemPageNum	Memory page number.
isLoadSection	Flag to indicate that the section is loadable.
Data	Buffer to hold data.

Comments

None.

Constraints

None.

See Also

None.

7.5 FnLoad

This is the Function pointer providing the abstraction to the loader's load component. All the loaders, which can be plugged into DSP/BIOS LINK, must have this function with correct signature.

Definition

```
typedef DSP_STATUS (*FnLoad) (IN  ProcessorId     procId,
                              IN  LoaderObject *  loaderObj,
                              IN  Uint32          argc,
                              IN  Char8 **        argv,
                              OUT Uint32 *        entryPt) ;
```

Fields

dspId	Target DSP identifier where the base image is to be loaded.
loaderObj	This object is used to receive arguments from PMGR.
argc	Number of arguments to be passed to the base image upon start.
argv	Arguments to be passed to DSP main application.
entryPt	Argument for returning entry address for the executable.

Comments

None.

Constraints

None.

See Also

None.

7.6 FnLoadSection

This is the Function pointer providing the abstraction to the loader's load-section component. All the loaders, which can be plugged into DSP/BIOS LINK, must have this function with correct signature.

Definition

```
typedef DSP_STATUS (*FnLoadSection) (IN  ProcessorId     procId,
                                     IN  LoaderObject *  loaderObj,
                                     IN  Uint32          sectId) ;
```

Fields

dspId	Target DSP identifier where the section is to be loaded.
loaderObj	This object is used to receive arguments from PMGR.
sectID	Identifier for section to load.

Comments

None.

Constraints

None.

See Also

None.

7.7 LoaderInterface

Interface functions exported by the Loader component.

Definition

```
typedef struct LoaderInterface_tag {
    FnLoad         load         ;
    FnLoadSection  loadSection  ;
} LoaderInterface ;
```

Fields

load	Function pointer providing the abstraction to the loader's load component.
loadSection	Function pointer providing the abstraction to the loader's loadSection component.

Comments

None.

Constraints

None.

See Also

None.

8 API Definition

The COFF Loader APIs are exposed to PMGR through a function table:

```
LoaderInterface Loader_COFF = {
    &COFF_Load,
    &COFF_LoadSection
} ;
```

8.1 COFF_Load

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

Syntax

```
DSP_STATUS COFF_Load (IN  ProcessorId     procId,
                      IN  LoaderObject *  loaderObj,
                      IN  Uint32          argc,
                      IN  Char8 **        argv,
                      OUT Uint32 *        entryPt) ;
```

Arguments

IN	ProcessorId	procId

Target DSP identifier where the base image must load.

IN	LoaderObject	loaderObj

This object is used to receive arguments from PMGR.

IN	Uint32	argc

Number of argument to pass to the base image upon start

IN	Char8 **	argv

Arguments to pass to the DSP main application

OUT	Uint32 *	entryPt

OUT argument for returning entry address for the executable.

Return Values

DSP_SOK	Base image successfully loaded
DSP_EFILE	Invalid base image
DSP_EACCESSDENIED	Not allowed to access the DSP
DSP_ECORRUPTFILE	File is not valid for this architecture.
DSP_EFAIL	General failure, unable to load image onto DSP
DSP_EINVALIDARG	Invalid procId argument.

Comments

Loads the Coff format file on the DSP. PMGR_PROC_Load calls this through the function pointer table. It also retrieves the start address of the base image and stores it in a private structure for future use (to be used in PMGR_PROC_Start()).

Constraints

procId must be a valid DSP processor ID.

baseImage must be a valid file identifier.

entryAddress must be a valid section identifier.

See Also

PMGR_PROC_Load

8.2 COFF_Debug

This function prints the debug information of COFF sub-component.

Syntax

```
DSP_STATUS COFF_Debug (CoffContext *   obj) ;
```

Arguments

IN	CoffContext	obj

The context object obtained through COFF_Initialize.

Return Values

DSP_SOK	Operation completed successfully

Comments

None.

Constraints

None.

See Also

None

8.3 COFF_LoadSection

This function loads a section from the DSP executable onto the DSP. PMGR_PROC_Load calls this through the function pointer table.

Syntax

```
DSP_STATUS COFF_LoadSection (IN  ProcessorId     procId,
                             IN  LoaderObject *  loaderObj,
                             IN  Uint32          sectId) ;
```

Arguments

IN	ProcessorId	ProcId
	Target DSP identifier where the base image must load.	
IN	LoaderObject	loaderObj
	This object is used to receive arguments from PMGR.	
IN	Uint32	SectId
	Identifier for section to load.	

Return Values

DSP_SOK	Base image successfully loaded
DSP_EFILE	Invalid base image
DSP_EACCESSDENIED	Not allowed to access the DSP
DSP_ECORRUPTFILE	File is not valid for this architecture.
DSP_EFAIL	General failure, unable to load image onto DSP
DSP_EINVALIDSECT	Invalid section name.
DSP_EINVALIDARG	Invalid procId argument.

Comments

Loads a section from the DSP executable onto the DSP. PMGR_PROC_Load calls this through the function pointer table.

Constraints

procId must be a valid DSP processor ID.

baseImage must be a valid file identifier.

entryAddress must be a valid section identifier.

See Also

PMGR_PROC_Load

8.4 COFF_Initialize

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

Syntax

```
DSP_STATUS COFF_Initialize (ProcessorId     procId,
                            Pstr            file,
                            DspArch         dspArch,
                            CoffContext *   obj) ;
```

Arguments

IN	ProcessorId	procId
	Processor Id	
IN	FileName	file
	Identifier for the file.	
IN	DspArch	dspArch
	Architecture of the DSP.	
OUT	Void **	obj
	OUT argument that contains the object to be passed in any subsequent call from this subcomponent.	

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File not found.
DSP_EMEMORY	Memory error

Comments

None.

Constraints

procId must be valid.

file must not be NULL.

obj must not be NULL.

See Also

COFF_Finalize

8.5 COFF_Finalize

This function releases the context object obtained through COFF_Initialize.

Syntax

```
DSP_STATUS COFF_Finalize (Pvoid  objCtx) ;
```

Arguments

IN Pvoid ObjCtx

The context object obtained through COFF_Initialize.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFILE	File not found.
DSP_EMEMORY	Operation failed due to memory error

Comments

None.

Constraints

objCtx must be valid.

See Also

COFF_Initialize

8.6 COFF_Read8

This function reads an Int8 from file.

Syntax

```
Int8 COFF_Read8 (KFileObject * fileObj) ;
```

Arguments

IN	KfileObject *	fileObj

File to read from.

Return Values

The read value.

Comments

None.

Constraints

fileObj must be valid.

See Also

None

8.7 COFF_Read32

This function reads an Int32 from file.

Syntax

```
Int32 COFF_Read32 (KFileObject * fileObj, Bool swap) ;
```

Arguments

IN	KfileObject *	fileObj

File to read from.

IN	Bool	swap

Flag to specify whether the bytes need to be swapped.

Return Values

The read value.

Comments

None.

Constraints

fileObj must be valid.

See Also

None

8.8 COFF_SeekToSectionHeader

This function repositions the file position indicator to the section header.

Syntax

```
DSP_STATUS COFF_SeekToSectionHeader (KFileObject * fileObj,
                                     Uint32        sectIndex,
                                     Bool          swap) ;
```

Arguments

IN	Pvoid	fileObj

Handle to the COFF file.

IN	Uint32	sectIndex

Section Index.

IN	Bool	swap

Flag to indicate that headers in this file are swapped.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be valid.

See Also

None.

8.9 COFF_IsSwapped

This function checks if the fields of headers are stored as byte swapped values.

Syntax

```
DSP_STATUS COFF_IsSwapped (KFileObject * fileObj,
                           DspArch       dspArch,
                           Bool *        isSwapped) ;
```

Arguments

IN	KFileObject *	FileObj	
	Handle to the COFF file.		
IN	DspArch	DspArch	
	IN argument to contain if the COFF headers in file are swapped.		
OUT	Bool *	IsSwapped	
	OUT argument to contain if the COFF headers in file are swapped.		

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be a valid pointer.

isSwapped must be a valid pointer.

See Also

COFF_IsSwapped_55x
COFF_IsSwapped_64x

8.10 COFF_IsValidFile

This function checks to indicate if the file data format is valid for the given architecture.

Syntax

```
DSP_STATUS COFF_IsValidFormat (KFileObject * fileObj,
                               DspArch       dspArch,
                               Bool *        isValid) ;
```

Arguments

IN	KFileObject *	FileObj

Handle to the COFF file.

IN	DspArch	DspArch

IN argument to contain if the COFF headers in file are swapped.

OUT	Bool *	IsValid

OUT argument to contain if the file data format is valid for the given architecture.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be a valid pointer.

isValid must be a valid pointer.

See Also

COFF_IsValidFile_55x
COFF_IsValidFile_64x

8.11 COFF_GetOptHeaderSize

This function gets the size of optional header in file. This function is used at many places to quickly seek to the desired field in file.

Syntax

```
DSP_STATUS COFF_GetOptHeaderSize (KFileObject * fileObj,
                                  Bool          swap,
                                  Int32 *       size) ;
```

Arguments

IN	KFileObject *	fileObj

Handle to the COFF file.

IN	Bool	swap

This flag specifies whether the bytes need to be swapped.

OUT	Int32 *	size

OUT argument to contain the optional header size.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be valid.

size must be valid.

See Also

None

8.12 COFF_GetSymTabDetails

This function gets the details associated to the symbol table i.e. number of symbols in the file and the offset of symbol table in file.

Syntax

```
DSP_STATUS COFF_GetSymTabDetails (KFileObject * fileObj,
                                  Bool          swap,
                                  Uint32 *      offsetSymTab,
                                  Uint32 *      numSymbols) ;
```

Arguments

IN	KFileObject *	fileObj

Handle to the COFF file.

IN	Bool	swap

Specifies whether the bytes need to be swapped.

OUT	Uint32 *	offsetSymTab

OUT argument to contain the offset of symbol table.

OUT	Uint32 *	numSymbols

OUT argument to contain the number of symbols.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be valid.

offsetSymTab must be valid.

numSymbols must be valid.

See Also

None

8.13 COFF_GetNumSections

This function gets the total number of sections in file.

Syntax

```
DSP_STATUS COFF_GetNumSections (KFileObject * fileObj,
                                Bool          swap,
                                Uint32 *      numSections) ;
```

Arguments

IN	KFileObject *	fileObj

Handle to the COFF file.

IN	Bool	swap

This flag specifies whether the bytes need to be swapped.

OUT	Uint32 *	numSections

OUT argument to contain the number of sections.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be valid.

numSections must be valid.

See Also

None

8.14 COFF_GetFileHeader

This function gets the File Header information. The caller should allocate memory for file header.

Syntax

```
DSP_STATUS COFF_GetFileHeader (CoffContext  *    obj,
                               CoffFileHeader *  fileHeader) ;
```

Arguments

IN	CoffContext *	obj

The context object obtained through COFF_Initialize.

OUT	CoffFileHeader *	fileHeader

OUT argument for containing file header information.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.
DSP_ERANGE	File seek operation failed.
DSP_EFILE	File format not supported.

Comments

None.

Constraints

obj must be valid.

fileHeader must be valid.

See Also

None

8.15 COFF_GetOptionalHeader

This function gets the COFF file's optional header. The caller should allocate memory for optional header.

Syntax

```
DSP_STATUS COFF_GetOptionalHeader (CoffContext   *  obj,
                                   CoffOptHeader *  optHeader) ;
```

Arguments

IN	CoffContext *	obj

The context object obtained through COFF_Initialize.

OUT	CoffOptHeader *	optHeader

OUT argument for containing optional header information.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.
DSP_ERANGE	File seek operation failed.
DSP_EFILE	File format not supported.

Comments

None.

Constraints

obj must be valid.

optHeader must be valid.

See Also

None

8.16 COFF_GetSectionHeader

This function gets the header information for a section. The caller should allocate memory for section header.

Syntax

```
DSP_STATUS COFF_GetSectionHeader (Uint32               sectId,
                                  CoffContext  *       obj,
                                  CoffSectionHeader *  sectHeader) ;
```

Arguments

IN	Uint32	sectId

Section index.

IN	CoffContext *	obj

The context object obtained through COFF_Initialize.

OUT	CoffSectionHeader *	sectHeader

OUT argument containing section header.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.
DSP_ERANGE	File seek operation failed.
DSP_EFILE	File format not supported.

Comments

None.

Constraints

obj must be valid.

sectHeader must be valid.

See Also

None

8.17 COFF_GetSectionData

This function gets the data associated with a section. Memory for buffer should be allocated prior to invoking this function.

Syntax

```
DSP_STATUS COFF_GetSectionData (Uint32         sectId,
                                CoffContext  * obj,
                                Char8 *        data) ;
```

Arguments

IN	Uint32	sectId

Section index.

IN	CoffContext *	obj

The context object obtained through COFF_Initialize.

OUT	Char8 *	data

OUT argument containing data buffer associated with the section.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.
DSP_ERANGE	File seek operation failed.
DSP_EFILE	File format not supported.

Comments

None.

Constraints

obj must be valid.

data must be valid.

See Also

None

8.18 COFF_GetString

This function gets the string from string table if required. This function checks if the 'str' argument is a valid string, if not, it looks up the string in string-table. Memory for string is allocated by this function.

Syntax

```
DSP_STATUS COFF_GetString (Char8 *        str,
                           CoffContext *  obj,
                           Char8 **       outStr) ;
```

Arguments

IN	Char8 *	str	
	Contains the string or the string offset.		
IN	CoffContext *	obj	
	The context object obtained through COFF_Initialize.		
OUT	Char8 **	outStr	
	OUT argument containing the string.		

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.
DSP_ERANGE	Binary file seek operation failed.
DSP_EFILE	File format not supported.

Comments

None.

Constraints

str must be valid.

obj must be valid.

outStr must be valid.

See Also

None

8.19 COFF_GetSymbolTable

This function gets the primary symbol entry for all the symbols in the coff file. The caller should allocate memory for the symbol table.

Syntax

```
DSP_STATUS COFF_GetSymbolTable (CoffContext  *     obj,
                                CoffSymbolEntry ** symTable,
                                Uint32           * numSymbols) ;
```

Arguments

IN CoffContext * Obj

The context object obtained through COFF_Initialize.

IN CoffSymbolEntry ** SymTable

OUT argument for holding the symbol table.

OUT Uint32 * NumSymbols

OUT argument for holding the actual number of distinct symbols present in the file.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_EINVALIDARG	Failure due to invalid argument.
DSP_ERANGE	Binary file seek operation failed.
DSP_EFILE	File format not supported.

Comments

None.

Constraints

obj must be valid.

symTable must be valid.

numSymbols must be valid.

See Also

None

8.20 COFF_FillArgsBuffer

This function fills up the specified buffer with arguments to be sent to DSP's "main" function.

Syntax

```
DSP_STATUS COFF_FillArgsBuffer (Uint32    argc,
                                Char8 **  argv,
                                Uint32    sectSize,
                                Uint32    loadAddr,
                                Uint32    wordsize,
                                Void *    argsBuf) ;
```

Arguments

`IN` `Uint32` `argc`

Number of arguments to be passed.

`IN` `Char8 **` `argv`

Argument strings to be passed.

`IN` `Uint32` `sectSize`

Size of the `'.args'` section obtained from the COFF file.

`IN` `Uint32` `LoadAddr`

Load address for the `'.args'` section.

`IN` `Uint32` `Wordsize`

Word size on the target DSP.

`IN` `Void *` `ArgsBuf`

Buffer to be filled with formatted `argc` and `argv`.

Return Values

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

Comments

None.

Constraints

`argc` must be greater than 0.

`argv` must be valid pointer.

`argsBuf` must be a valid pointer.

`sizeBuf` must be a valid pointer.

See Also

None

8.21 COFF_IsSwapped_55x

Checks if the fields of headers are stored as byte swapped values in 55x file format.

Syntax

```
DSP_STATUS COFF_IsSwapped_55x (KFileObject * fileObj,
                               Bool * isSwapped) ;
```

Arguments

IN	KfileObject	fileObj

Handle to the COFF file.

IN	Bool	isSwapped

OUT argument to contain if the COFF headers in file are swapped.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be a valid pointer.

isSwapped must be a valid pointer.

See Also

COFF_IsSwapped
COFF_IsSwapped_64x

8.22 COFF_IsValidFile_55x

Checks to indicate if the file data format is valid for 55x architecture.

Syntax

```
DSP_STATUS COFF_IsValidFile_55x (KFileObject * fileObj,
                                 Bool * isValid) ;
```

Arguments

IN	KfileObject	fileObj

Handle to the COFF file.

IN	Bool	isValid

OUT argument to contain if the file has a valid COFF 55x format.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

`fileObj` must be a valid pointer.

`isValid` must be a valid pointer.

See Also

COFF_IsValidFile
COFF_IsValidFile_64x

8.23 COFF_IsSwapped_64x

Checks if the fields of headers are stored as byte swapped values in 64x file format.

Syntax

```
DSP_STATUS COFF_IsSwapped_64 (KFileObject * fileObj,
                              Bool * isSwapped) ;
```

Arguments

IN	KfileObject	FileObj

Handle to the COFF file.

OUT	Bool	IsSwapped

OUT argument to contain if the COFF headers in file are swapped.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

`fileObj` must be a valid pointer.

`isSwapped` must be a valid pointer.

See Also

COFF_IsSwapped
COFF_IsSwapped_55x

8.24 COFF_IsValidFile_64x

Checks to indicate if the file data format is valid for 55x architecture.

Syntax

```
DSP_STATUS COFF_IsValidFile_64x (KFileObject * fileObj,
                                 Bool * isValid) ;
```

Arguments

IN	KfileObject	fileObj

Handle to the COFF file.

IN	Bool	isValid

OUT argument to contain if the file has a valid COFF 64x format.

Return Values

DSP_SOK	Operation completed successfully
DSP_EFAIL	General Failure.
DSP_RANGE	Seek error in file.

Comments

None.

Constraints

fileObj must be a valid pointer.

isValid must be a valid pointer.

See Also

COFF_IsValidFile
COFF_IsValidFile_55x

- The scalability issue of 55x and 64x is still under consideration.
- The path of the architecture specific files in COFF loader is still not finalized.
- COFF_GetSymbolTable is not used anywhere in the design but it may be useful in the implementation of BRIDGE over LINK.