XR/station Project
XLO File Format Specification
Revision 1.0, May 29, 2024

Table of Contents

1. Overview

The XR/SDK Linkable Object **(XLO)** file format is the native object file format of the XR/SDK suite of tools. It is the only file format emitted by the XR/ASM assembler, and is the only file format accepted by the XR/LINK linker. It is suitable for use as an intermediate object code format, as an executable file format, as a static library format, and as a dynamic library format.

XLO is a portable format, with current support for the RISC XR/17032 and CISC fox32 architectures, and with planned support for the Aphelion 64-bit architecture.

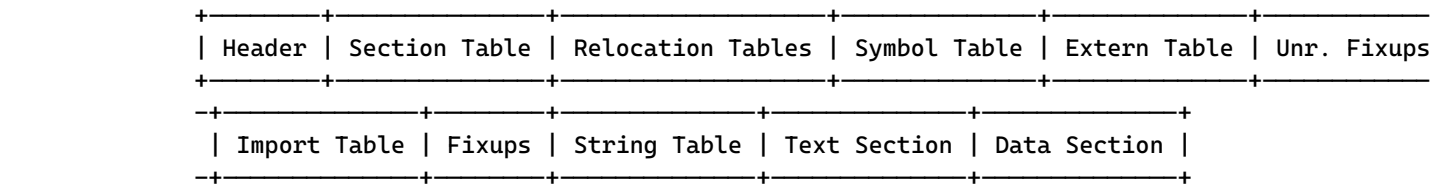
The format is simple, and architecture-specific details are limited to the definition of new relocation entry types. It is designed for flexibility, with the ability to specify arbitrary sections. The tables are laid out for rapid iteration of relevant entries during the process of load-time program relocation and dynamic linking, and an “optimistic” relocation scheme helps avoid both of these costs entirely.

2. Module Format

The overall format of an **XLO** module file is a header, followed by a number of tables linked together via file pointers (i.e. byte offsets into the file), and finally the data for each section.

The metadata contained within the file, that is, the header and the tables, are referred to collectively as the “head” of the file.

An maximal module file might have the following layout:



The “head length” of this file is the length of all of the contents up to the end of the last metadata; in this case, the string table.

3. Header

```
STRUCT XloHeader
    Magic : ULONG,

    SymbolTableOffset : ULONG,
    SymbolCount : ULONG,

    StringTableOffset : ULONG,
    StringTableSize : ULONG,

    TargetArchitecture : ULONG,

    HeadLength : ULONG,

    ImportTableOffset : ULONG,

    Flags : ULONG,
    Timestamp : ULONG,

    SectionTableOffset : ULONG,
    ExternTableOffset : ULONG,

    ExternCount : ULONG,
    SectionCount : UINT,
    ImportCount : UINT,
END

STRUCT XloHeaderExtended
    Hdr : XloHeader,

    UnresolvedFixupTableOffset : ULONG,
    UnresolvedFixupCount : ULONG,
END
```

The header of an **XLO** file contains general information about the module file, and provides information required to find and parse the tables of metadata. There are two variants of the header, “normal” and “extended”. These vary only by the extended header featuring two extra fields. The extended header is present within “fragment” modules, that is, modules that have the **XLO_FILE_FRAGMENT** flag (bit 0) set within the Flags field of the normal header.

In future revisions, extra fields may be added beyond the end of the extended header, but the header’s length *must* remain 64-bit aligned.

3.1. Magic

The 32-bit magic number in the normal header should read 0x6174737F.

3.2. SymbolTableOffset, SymbolCount

SymbolTableOffset contains the file pointer of the table which describes the symbols exposed by the module. SymbolCount contains the number of entries within that table. If SymbolCount is zero, SymbolTableOffset has undefined meaning.

3.3. StringTableOffset, StringTableSize

StringTableOffset contains the file pointer of the “string table”, which is the hunk of all null-terminated ASCII strings used by metadata within the module. StringTableSize

contains its length, up to (and including) the null terminator of the final string. If StringTableSize is zero, StringTableOffset has undefined meaning.

3.4. TargetArchitecture

This field contains the 32-bit "architecture code" indicating which instruction set the code within the module is for. Currently defined codes are:

Code	Architecture
00000000	Unknown
00000001	XR/17032
00000002	Fox32
00000003	Aphelion
00000004	AMD64

3.5. HeadLength

This field contains the length in bytes of all of the metadata for the module. It must therefore be grouped together at the beginning of the file to form a region known as the "head" that precedes all section data.

3.6. ImportTableOffset, ImportCount

ImportTableOffset contains the file pointer of the "import table", a flat array of entries which describe the dynamically linked libraries that are depended upon by this module. ImportCount contains the 16-bit count (range [0, 65535]) of entries in this table. If the module is a fragment (**XLO_FILE_FRAGMENT** is set in the Flags field), ImportCount must be zero. When ImportCount is zero, the meaning of ImportTableOffset is undefined.

3.7. Flags

This field contains up to 32 flags indicating characteristics of the module file. Currently defined flags are:

Bit	Name	Meaning
0	**XLO_FILE_FRAGMENT**	This file is a fragment; it has an extended header and is not yet suitable for relocation or dynamic linking. These files are produced directly by the assembler.

3.8. Timestamp

This field contains a 32-bit Unix Epoch timestamp (in seconds) of when the module file was encoded to disk. It is intended primarily to provide a unique versioning among multiple versions of the same dynamic library. When a dynamic library is linked against, its Timestamp field is captured in the import table entry. Mismatched timestamps indicate to the runtime dynamic linker that the library was updated, and that any modules that reference the old version must be fixed up.

3.9. SectionTableOffset, SectionCount

SectionTableOffset contains the file pointer of the "section table", a flat array of "section headers" that describe the sections contained within the module file. SectionCount contains the 8-bit number (range [0, 255]) of entries in this table. When it is zero, the meaning of SectionTableOffset is undefined.

SectionCount can physically contain a 16-bit count, but other fields within the format limit the number of sections in a single module to 255.

3.10. ExternTableOffset, ExternCount

ExternTableOffset contains the file pointer of the “extern table”, a flat array that describes all required symbols that reside in other modules. ExternCount is the 16-bit number (range [0, 65535]) of entries in this table. When it is zero, the meaning of ExternTableOffset is undefined.

3.11. UnresolvedFixupTableOffset, UnresolvedFixupCount

These two entries reside in the extended header and therefore only exist in fragment modules. UnresolvedFixupTableOffset contains the file pointer of the “unresolved fixup table”, a flat array of relocation entries that depend on the value of unresolved extern symbols in order to be processed. UnresolvedFixupCount contains the number of entries in this table. If it is zero, the meaning of UnresolvedFixupTableOffset is undefined.

4. Symbol Table

```
STRUCT XloSymbolEntry
    SectionIndex : UBYTE,
    Type : UBYTE,
    Flags : UBYTE,
    Padding : UBYTE,
    SectionOffset : ULONG,
    NameOffset : ULONG,
END
```

The symbol table is an array of symbol entries, each representing a named value that is exposed by the module. This structure is essential for linking (both static and dynamic) and debugging (for stack traces, etc). A symbol normally corresponds to a function, variable, or data structure defined in a high-level language like Jackal.

4.1. SectionIndex

The 8-bit index (range [0, 255]) into the section table of the section that this symbol resides in; i.e. the section that the SectionOffset field is relative to.

4.2. Type

The 8-bit type code indicating properties of the symbol. Currently defined types are:

Code	Name	Meaning
0x01	**GLOBAL**	This symbol is visible to other modules in a statically linked compilation unit, but will not be included in the symbol table of a final executable or dynamic library.
0x02	**EXPORT**	This symbol is visible to other modules in both a statically and dynamically linked unit. Is included in a final symbol table after linking.

4.3. Flags

Up to 8 flags indicating characteristics of the symbol. No symbol flags are currently defined.

4.4. SectionOffset

The offset within the section at which the symbol resides.

4.5. NameOffset

The offset from the base of the string table at which the null-terminated ASCII symbol name resides.

5. Import Table

```
STRUCT XloImportEntry
    NameOffset : ULONG,
    ExpectedTimestamp : ULONG,
    FixupTableOffset : ULONG,
    FixupCount : ULONG,
END
```

The import table is the array of entries that describe the dynamic libraries upon which this module depends at runtime.

5.1. NameOffset

The offset from the base of the string table at which the null-terminated ASCII dynamic library name resides.

5.2. ExpectedTimestamp

This field contains a 32-bit Unix Epoch timestamp (in seconds), captured from the Timestamp field of the dynamic library's header. It is intended primarily to provide a unique versioning among multiple versions of the same dynamic library. When a dynamic library is linked against, its Timestamp field is captured here. Mismatched timestamps indicate to the runtime dynamic linker that the library was updated, and that this module must be fixed up.

5.3. FixupTableOffset, FixupCount

FixupTableOffset contains the file pointer of a "fixup table", containing all of the relocations that must be performed at runtime should this dynamic library have a mismatched version, or fail to load at its preferred base address. FixupCount contains the number of entries in this table.

6. Relocation and Fixup Tables

There are several "relocation tables" within the **XLO** format:

- The per-section relocation tables, containing all of the "internal" relocations that must be performed in all sections if that section is moved in the virtual address space.
- The per-import fixup tables, containing all of the "external" relocations that must be performed if that imported dynamic library is of an unexpected version, or if it fails to load at its preferred base address.
- The unresolved fixup table, containing all of the external relocations that must be performed against the value of extern symbols that are still of totally unknown origin.

The entries of each kind of table share a format, which they each use somewhat differently:

```
STRUCT XloRelocEntry
    SectionOffset : ULONG,
    ExternIndex : UINT,
    Type : UBYTE,
    SectionIndex : UBYTE,
END
```

6.1. SectionOffset

Indicates the offset within the "target section" of the pointer that must be relocated.

6.2. ExternIndex

Indicates the 16-bit index (range [0, 65535]) of the entry within the extern table that describes the external symbol this relocation relies upon. This field has no meaning and is unused if this is an internal (i.e. per-section table) relocation.

6.3. Type

Indicates the 8-bit type code (range [0, 255]) of the pointer that must be relocated. The currently defined types are:

Code	Name	Meaning
0x01	**PTR**	32 or 64-bit pointer, depending on the bitness of the module's target architecture.
0x02	**XR17032_ABSJ**	An XR/17032 absolute jump instruction.
0x03	**XR17032_LA**	An XR/17032 **LA** pseudo-instruction.
0x04	**XR17032_FAR_INT**	An XR/17032 far-int access pseudo-instruction.
0x05	**XR17032_FAR_LONG**	An XR/17032 far-long access pseudo-instruction.
0x06	**FOX32_CALL**	A fox32 **CALL** instruction.

6.4. SectionIndex

The 8-bit index (range [0, 255]) into the section table of the "target section" that this relocation modifies; i.e., the section that the SectionOffset is relative to.

7. Extern Table

```
STRUCT XloExternEntry
    NameOffset : ULONG,
    Type : UBYTE,
    Padding : UBYTE,
    ImportIndex : UINT,
    Padding2 : ULONG,
    Padding3 : ULONG,
END
```

The extern table is an array of "external symbol" entries, each representing a named value that is external to, but depended upon by the module. This structure is essential for linking. An extern normally corresponds to a function, variable, or data structure defined in a high-level language like Jackal.

7.1. NameOffset

The offset from the base of the string table at which the null-terminated ASCII name of the external symbol resides.

7.2. Type

The 8-bit type code indicating properties of the extern. Currently defined types are:

Type	Name	Meaning
1	**UNRESOLVED**	This external symbol is completely unresolved.
2	**IMPORTED**	This external symbol resides in a known dynamic library.

7.3. ImportIndex

The 16-bit index (range [0, 65535]) of the import table entry that describes the dynamic library this external symbol resides in. If this external symbol is not of type **IMPORTED**, this field has no meaning.

8. Section Table

```
STRUCT XloSectionHeader

#IF ( == BITS 64 )
    VirtualAddress : UQUAD,
#ELSE
    VirtualAddress : ULONG,
    Reserved : ULONG,
#END

    NameOffset : ULONG,
    FileOffset : ULONG,
    DataLength : ULONG,
    RelocTableOffset : ULONG,
    RelocCount : ULONG,
    Flags : ULONG,
END
```

The section table is a flat array of “section headers” that describe hunks of data and code contained by this module. The file pointer of the section table must be 64-bit aligned as the section header contains a 64-bit field.

8.1. VirtualAddress

VirtualAddress contains the “link-time” base address to which the section has been placed; that is, the “assumed” address that all pointers to the section have been offsetted by. If at runtime the section cannot be placed at this address, internal relocations for this module (and external fixups for other modules that may be dynamically linked to it) must be performed.

This field is either 32 bits or 64 bits depending on the bitness of the target architecture. This allows sections to be located anywhere within a 64-bit address space, but their sizes are still limited to 4GB each, due to pervasive use of 32-bit section offsets. For 32-bit modules, the space where the upper 32 bits of the virtual address would be should be zero, to ensure compatibility with 64-bit tools.

8.2. NameOffset

The offset from the base of the string table at which the null-terminated ASCII name of the section resides.

8.3. FileOffset

The file pointer of the section contents within the module.

8.4. DataLength

The length of the section contents.

8.5. RelocTableOffset and RelocCount

RelocTableOffset contains the file pointer of the section’s relocation table, containing all of the internal relocations that must be performed at runtime should this section fail to be placed at its preferred virtual address. RelocCount contains the number of entries within this table.

8.6. Flags

Up to 32 flags that indicate characteristics of the section. Currently defined flags are:

Bit	Name	Meaning
0	**XLO_SECTION_ZERO**	The section has no on-disk data and should be zeroed out in memory at load time (i.e., it is a “bss” section).
1	**XLO_SECTION_CODE**	The section contains code and should be mapped as executable.
2	**XLO_SECTION_MAP**	The section has in-memory presence at load time. If this isn’t set, it only has on-disk data such as debug information.