The SFrame Format

Version 2 (Errata 1)

27 July 2025

Indu Bhagat

Copyright © 2021-2025 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU General Public License, Version 3 or any later version published by the Free Software Foundation. A copy of the license is included in the section entitled “GNU General Public License”.

Table of Contents

1 Introduction 1
1.1 Overview 1
1.2 Changes from Version 1 to Version 2 1

2 SFrame Section 3
2.1 SFrame Preamble 3
2.1.1 SFrame Magic Number and Endianness 3
2.1.2 SFrame Version 3
2.1.3 SFrame Flags 4
2.2 SFrame Header 4
2.2.1 SFrame ABI/arch Identifier 6
2.3 SFrame FDE 6
2.3.1 The SFrame FDE Info Word 8
2.3.2 The SFrame FDE Types 8
2.3.3 The SFrame FRE Types 9
2.4 SFrame FRE 9
2.4.1 The SFrame FRE Info Word 10

3 ABI/arch-specific Definition 12
3.1 AMD64 12
3.2 AArch64 12
3.3 s390x 13

Appendix A Generating Stack Traces using SFrame 15

Index 17

1 Introduction

1.1 Overview

The SFrame stack trace information is provided in a loaded section, known as the `.sframe` section. When available, the `.sframe` section appears in segment of type PT_GNU_SFRAME. An ELF SFrame section will have the type SHT_GNU_SFRAME.

The SFrame format is currently supported only for select ABIs, namely, AMD64, AAPCS64, and s390x.

A portion of the SFrame format follows an unaligned on-disk representation. Some data structures, however, (namely the SFrame header and the SFrame function descriptor entry) have elements at their natural boundaries. All data structures are packed, unless otherwise stated.

The contents of the SFrame section are stored in the target endianness, i.e., in the endianness of the system on which the section is targeted to be used. An SFrame section reader may use the magic number in the SFrame header to identify the endianness of the SFrame section.

Addresses in this specification are expressed in bytes.

The rest of this specification describes the current version of the format, `SFRAME_VERSION_2`, in detail. Additional sections outline the major changes made to each previously published version of the SFrame stack trace format.

The associated API to decode, probe and encode the SFrame section, provided via `libsframe`, is not accompanied here at this time. This will be added later.

This document is intended to be in sync with the C code in `sframe.h`. Please report discrepancies between the two, if any.

1.2 Changes from Version 1 to Version 2

The following is a list of the changes made to the SFrame stack trace format since Version 1 was published.

- Add an unsigned 8-bit integral field to the SFrame function descriptor entry to encode the size of the repetitive code blocks. Such code blocks, e.g, pltN entries, use an SFrame function descriptor entry of type SFRAME_FDE_TYPE_PCMASK.
- Add an unsigned 16-bit integral field to the SFrame function descriptor entry to serve as padding. This helps ensure natural alignment for the members of the data structure.
- The above two imply that each SFrame function descriptor entry has a fixed size of 20 bytes instead of its size of 17 bytes in SFrame format version 1.
- [Errata 1] Add a new flag SFRAME_F_FDE_FUNC_START_PCREL, as an erratum to SFrame Version 2, to indicate the encoding of the SFrame FDE function start address field:
  - if set, `sfde_func_start_address` field contains the offset in bytes to the start PC of the associated function from the field itself.
  - if unset, `sfde_func_start_address` field contains the offset in bytes to the start PC of the associated function from the start of the SFrame section.

- [Errata 1] Add a new ABI/arch identifier SFRAME_ABI_S390X_ENDIAN_BIG for the s390 architecture (64-bit) s390x ABI. Other s390x-specific backward compatible changes including the following helper definitions have been incrementally added to SFrame version 2 only:
  - SFRAME_S390X_SP_VAL_OFFSET: SP value offset from CFA.
  - SFRAME_V2_S390X_OFFSET_IS_REGNUM: Test whether FP/RA offset is an encoded DWARF register number.
  - SFRAME_V2_S390X_OFFSET_ENCODE_REGNUM: Encode a DWARF register number as an FP/RA offset.
  - SFRAME_V2_S390X_OFFSET_DECODE_REGNUM: Decode a DWARF register number from an FP/RA offset.
  - SFRAME_FRE_RA_OFFSET_INVALID: Invalid RA offset value (like SFRAME_CFA_FIXED_RA_INVALID). Used on s390x as padding offset to represent FP without RA saved.
  - SFRAME_S390X_CFA_OFFSET_ADJUSTMENT: CFA offset (from CFA base register) adjustment value. Used to enable use of 8-bit SFrame offsets on s390x.
  - SFRAME_S390X_CFA_OFFSET_ALIGNMENT_FACTOR: CFA offset alignment factor. Used to scale down the CFA offset to improve the use of 8-bit SFrame offsets.
  - SFRAME_V2_S390X_CFA_OFFSET_ENCODE: Encode CFA offset (i.e., apply CFA offset adjustment and then scale down by CFA offset alignment factor).
  - SFRAME_V2_S390X_CFA_OFFSET_DECODE: Decode CFA offset (i.e., scale up by CFA offset alignment factor and then revert CFA offset adjustment).
- [Errata 1] An ELF SFrame section has the type SHT_GNU_SFRAME.

SFrame version 1 is now obsolete and should not be used.

2 SFrame Section

The SFrame section consists of an SFrame header, starting with a preamble, and two other sub-sections, namely the SFrame function descriptor entry (SFrame FDE) sub-section, and the SFrame frame row entry (SFrame FRE) sub-section.

2.1 SFrame Preamble

The preamble is a 32-bit packed structure; the only part of the SFrame section whose format cannot vary between versions.

```
typedef struct sframe_preamble
{
  uint16_t sfp_magic;
  uint8_t sfp_version;
  uint8_t sfp_flags;
} ATTRIBUTE_PACKED sframe_preamble;
```

Every element of the SFrame preamble is naturally aligned.

All values are stored in the endianness of the target system for which the SFrame section is intended. Further details:

Offset	Type	Name	Description
0x00	`uint16_t`	`sfp_magic`	The magic number for SFrame section: 0xdee2. Defined as a macro `SFRAME_MAGIC`.
0x02	`uint8_t`	`sfp_version`	The version number of this SFrame section. See Section 2.1.2 [SFrame Version], page 3, for the set of valid values. Current version is `SFRAME_VERSION_2`.
0x03	`uint8_t`	`sfp_flags`	Flags (section-wide) for this SFrame section. See Section 2.1.3 [SFrame Flags], page 4, for the set of valid values.

2.1.1 SFrame Magic Number and Endianness

SFrame sections are stored in the target endianness of the system that consumes them. A consumer library reading or writing SFrame sections should detect foreign-endianness by inspecting the SFrame magic number in the `sfp_magic` field in the SFrame header. It may then provide means to endian-flip the SFrame section as necessary.

2.1.2 SFrame Version

The version of the SFrame format can be determined by inspecting `sfp_version`. The following versions are currently valid:

Version Name	Number	Description
`SFRAME_VERSION_1`	1	First version, obsolete.
`SFRAME_VERSION_2`	2	Current version, under development.

This document describes `SFRAME_VERSION_2`.

2.1.3 SFrame Flags

The preamble contains bitflags in its `sfp_flags` field that describe various section-wide properties.

The following flags are currently defined.

Flag	Version	Value	Meaning
`SFRAME_F_FDE_SORTED`	All	0x1	Function Descriptor Entries are sorted on PC.
`SFRAME_F_FRAME_POINTER`	All	0x2	All functions in the object file preserve frame pointer.
`SFRAME_F_FDE_FUNC_START_PCREL`	2	0x4	The `sfde_func_start_address` field in the SFrame FDE is an offset in bytes to the function's start address, from the field itself. If unset, the `sfde_func_start_address` field in the SFrame FDE is an offset in bytes to the function's start address, from the start of the SFrame section.

The purpose of SFRAME_F_FRAME_POINTER flag is to facilitate stack tracers to reliably fallback on the frame pointer based stack tracing method, if SFrame information is not present for some function in the SFrame section.

Further flags may be added in future. Bits corresponding to the currently undefined flags must be set to zero.

2.2 SFrame Header

The SFrame header is the first part of an SFrame section. It begins with the SFrame preamble. All parts of it other than the preamble (see Section 2.1 [SFrame Preamble], page 3) can vary between SFrame file versions. It contains things that apply to the section as a whole, and offsets to the various other sub-sections defined in the format. As with the rest of the SFrame section, all values are stored in the endianness of the target system.

The two sub-sections tile the SFrame section: each section runs from the offset given until the start of the next section. An explicit length is given for the last sub-section, the SFrame Frame Row Entry (SFrame FRE) sub-section.

```
typedef struct sframe_header
{
  sframe_preamble sfh_preamble;
  uint8_t sfh_abi_arch;
  int8_t sfh_cfa_fixed_fp_offset;
  int8_t sfh_cfa_fixed_ra_offset;
  uint8_t sfh_auxhdr_len;
  uint32_t sfh_num_fdes;
  uint32_t sfh_num_fres;
  uint32_t sfh_fre_len;
  uint32_t sfh_fdeoff;
  uint32_t sfh_freoff;
```

```
} ATTRIBUTE_PACKED sframe_header;
```

Every element of the SFrame header is naturally aligned.

The sub-section offsets, namely `sfh_fdeoff` and `sfh_freoff`, in the SFrame header are relative to the *end* of the SFrame header; they are each an offset in bytes into the SFrame section where the SFrame FDE sub-section and the SFrame FRE sub-section respectively start.

The SFrame section contains `sfh_num_fdes` number of fixed-length array elements in the SFrame FDE sub-section. Each array element is of type SFrame function descriptor entry; each providing a high-level function description for the purpose of stack tracing. More details in a subsequent section. See Section 2.3 [SFrame Function Descriptor Entries], page 6.

Next, the SFrame FRE sub-section, starting at offset `sfh_fre_off`, describes the stack trace information for each function, using a total of `sfh_num_fres` number of variable-length array elements. Each array element is of type SFrame frame row entry. See Section 2.4 [SFrame Frame Row Entries], page 9.

SFrame header allows specifying explicitly the fixed offsets from CFA, if any, from which FP or RA may be recovered. For example, in AMD64, the stack offset of the return address is `CFA - 8`. Since these offsets are expected to be in close vicinity to the CFA in most ABIs, `sfh_cfa_fixed_fp_offset` and `sfh_cfa_fixed_ra_offset` are limited to signed 8-bit integers.

The SFrame format has made some provisions for supporting more ABIs/architectures in the future. One of them is the concept of the auxiliary SFrame header. Bytes in the auxiliary SFrame header may be used to convey further ABI-specific information. The `sframe_header` structure provides an unsigned 8-bit integral field to denote the size (in bytes) of an auxiliary SFrame header. The auxiliary SFrame header follows right after the `sframe_header` structure. As for the calculation of the sub-section offsets, namely `sfh_fdeoff` and `sfh_freoff`, the *end* of SFrame header must be the end of the auxiliary SFrame header, if the latter is present.

Putting it all together:

Offset	Type	Name	Description
0x00	`sframe_preamble`	`sfh_preamble`	The SFrame preamble. See Section 2.1 [SFrame Preamble], page 3.
0x04	`uint8_t`	`sfh_abi_arch`	The ABI/arch identifier. See Section 2.2.1 [SFrame ABI/arch Identifier], page 6.
0x05	`int8_t`	`sfh_cfa_fixed_fp_offset`	The CFA fixed FP offset, if any.
0x06	`int8_t`	`sfh_cfa_fixed_ra_offset`	The CFA fixed RA offset, if any.
0x07	`uint8_t`	`sfh_auxhdr_len`	Size in bytes of the auxiliary header that follows the `sframe_header` structure.

0x08	`uint32_t`	`sfh_num_fdes`	The number of SFrame FDEs in the section.
0x0c	`uint32_t`	`sfh_num_fres`	The number of SFrame FREs in the section.
0x10	`uint32_t`	`sfh_fre_len`	The length in bytes of the SFrame FRE sub-section.
0x14	`uint32_t`	`sfh_fdeoff`	The offset in bytes to the SFrame FDE sub-section.
0x18	`uint32_t`	`sfh_freoff`	The offset in bytes to the SFrame FRE sub-section.

2.2.1 SFrame ABI/arch Identifier

SFrame header identifies the ABI/arch of the target system for which the executable and hence, the stack trace information contained in the SFrame section, is intended. There are currently three identifiable ABI/arch values in the format.

ABI/arch Identifier	**Value**	**Description**
`SFRAME_ABI_AARCH64_ENDIAN_BIG`	1	AARCH64 big-endian
`SFRAME_ABI_AARCH64_ENDIAN_LITTLE`	2	AARCH64 little-endian
`SFRAME_ABI_AMD64_ENDIAN_LITTLE`	3	AMD64 little-endian
`SFRAME_ABI_S390X_ENDIAN_BIG`	4	s390x big-endian

The presence of an explicit identification of ABI/arch in SFrame may allow stack trace generators to make certain ABI/arch-specific decisions.

2.3 SFrame FDE

The SFrame function descriptor entry sub-section is an array of the fixed-length SFrame function descriptor entries (SFrame FDEs). Each SFrame FDE is a packed structure which contains information to describe a function's stack trace information at a high-level.

The array of SFrame FDEs is sorted on the `sfde_func_start_address` if the SFrame section header flag `sfp_flags` has `SFRAME_F_FDE_SORTED` set. Typically (as is the case with GNU ld) a linked object or executable will have the `SFRAME_F_FDE_SORTED` set. This makes the job of a stack tracer easier as it may then employ binary search schemes to look for the pertinent SFrame FDE.

```
typedef struct sframe_func_desc_entry
{
  int32_t sfde_func_start_address;
```

```
  uint32_t sfde_func_size;
  uint32_t sfde_func_start_fre_off;
  uint32_t sfde_func_num_fres;
  uint8_t sfde_func_info;
  uint8_t sfde_func_rep_size;
  uint16_t sfde_func_padding2;
} ATTRIBUTE_PACKED sframe_func_desc_entry;
```

Every element of the SFrame function descriptor entry is naturally aligned.

`sfde_func_start_fre_off` is the offset to the first SFrame FRE for the function. This offset is relative to the *end of the SFrame FDE* sub-section (unlike the sub-section offsets in the SFrame header, which are relative to the *end* of the SFrame header).

`sfde_func_info` is the SFrame FDE "info word", containing information on the FRE type and the FDE type for the function See Section 2.3.1 [The SFrame FDE Info Word], page 8.

Apart from the `sfde_func_padding2`, the SFrame FDE has some currently unused bits in the SFrame FDE info word, See Section 2.3.1 [The SFrame FDE Info Word], page 8, that may be used for the purpose of extending the SFrame file format specification for future ABIs.

Following table describes each component of the SFrame FDE structure:

Offset	**Type**	**Name**	**Description**
0x00	`int32_t`	`sfde_func_start_address`	Signed 32-bit integral field denoting the virtual memory address of the described function, for which the SFrame FDE applies. If the flag `SFRAME_F_FDE_FUNC_START_PCREL`, See Section 2.1.3 [SFrame Flags], page 4, in the SFrame header is set, the value encoded in the `sfde_func_start_address` field is the offset in bytes to the function's start address, from the SFrame `sfde_func_start_address` field.
0x04	`uint32_t`	`sfde_func_size`	Unsigned 32-bit integral field specifying the size of the function in bytes.
0x08	`uint32_t`	`sfde_func_start_fre_off`	Unsigned 32-bit integral field specifying the offset in bytes of the function's first SFrame FRE in the SFrame section.
0x0c	`uint32_t`	`sfde_func_num_fres`	Unsigned 32-bit integral field specifying the total number of SFrame FREs used for the function.

0x10	`uint8_t`	`sfde_func_info`	Unsigned 8-bit integral field specifying the SFrame FDE info word. See Section 2.3.1 [The SFrame FDE Info Word], page 8.
0x11	`uint8_t`	`sfde_func_rep_size`	Unsigned 8-bit integral field specifying the size of the repetitive code block for which an SFrame FDE of type SFRAME_FDE_TYPE_PCMASK is used. For example, in AMD64, the size of a pltN entry is 16 bytes.
0x12	`uint16_t`	`sfde_func_padding2`	Padding of 2 bytes. Currently unused bytes.

2.3.1 The SFrame FDE Info Word

The info word is a bitfield split into three parts. From MSB to LSB:

Bit offset	Name	Description
7–6	`unused`	Unused bits.
5	`pauth_key`	(For AARCH64) Specify which key is used for signing the return addresses in the SFrame FDE. Two possible values: SFRAME_AARCH64_PAUTH_KEY_A (0), or SFRAME_AARCH64_PAUTH_KEY_B (1). Ununsed in AMD64.
4	`fdetype`	Specify the SFrame FDE type. Two possible values: SFRAME_FDE_TYPE_PCMASK (1), or SFRAME_FDE_TYPE_PCINC (0). See Section 2.3.2 [The SFrame FDE Types], page 8.
0–3	`fretype`	Choice of three SFrame FRE types. See Section 2.3.3 [The SFrame FRE Types], page 9.

2.3.2 The SFrame FDE Types

The SFrame format defines two types of FDE entries. The choice of which SFrame FDE type to use is made based on the instruction patterns in the relevant program stub.

An SFrame FDE of type `SFRAME_FDE_TYPE_PCINC` is an indication that the PCs in the FREs should be treated as increments in bytes. This is used fo the the bulk of the executable code of a program, which contains instructions with no specific pattern.

In contrast, an SFrame FDE of type `SFRAME_FDE_TYPE_PCMASK` is an indication that the PCs in the FREs should be treated as masks. This type is useful for the cases where a small pattern of instructions in a program stub is used repeatedly for a specific functionality. Typical usecases are pltN entries and trampolines.

Name of SFrame FDE type	Value	Description
SFRAME_FDE_TYPE_PCINC	0	Stacktracers perform a (PC >= FRE_START_ADDR) to look up a matching FRE.
SFRAME_FDE_TYPE_PCMASK	1	Stacktracers perform a (PC % REP_BLOCK_SIZE >= FRE_START_ADDR) to look up a matching FRE. REP_BLOCK_SIZE is the size in bytes of the repeating block of program instructions and is encoded via `sfde_func_rep_size` in the SFrame FDE.

2.3.3 The SFrame FRE Types

A real world application can have functions of size big and small. SFrame format defines three types of SFrame FRE entries to effeciently encode the stack trace information for such a variety of function sizes. These representations vary in the number of bits needed to encode the start address offset in the SFrame FRE.

The following constants are defined and used to identify the SFrame FRE types:

Name	Value	Description
`SFRAME_FRE_TYPE_ADDR1`	0	The start address offset (in bytes) of the SFrame FRE is an unsigned 8-bit value.
`SFRAME_FRE_TYPE_ADDR2`	1	The start address offset (in bytes) of the SFrame FRE is an unsigned 16-bit value.
`SFRAME_FRE_TYPE_ADDR4`	2	The start address offset (in bytes) of the SFrame FRE is an unsigned 32-bit value.

A single function must use the same type of SFrame FRE throughout. The identifier to reflect the chosen SFrame FRE type is stored in the `fretype` bits in the SFrame FDE info word, See Section 2.3.1 [The SFrame FDE Info Word], page 8.

2.4 SFrame FRE

The SFrame frame row entry sub-section contains the core of the stack trace information. An SFrame frame row entry (FRE) is a self-sufficient record containing SFrame stack trace information for a range of contiguous (instruction) addresses, starting at the specified offset from the start of the function.

Each SFrame FRE encodes the stack offsets to recover the CFA, FP and RA (where applicable) for the respective instruction addresses. To encode this information, each SFrame FRE is followed by S*N bytes, where:

- `S` is the size of a stack offset for the FRE, and
- `N` is the number of stack offsets in the FRE

The entities `S`, `N` are encoded in the SFrame FRE info word, via the `fre_offset_size` and the `fre_offset_count` respectively. More information about the precise encoding and range of values for `S` and `N` is provided later in the See Section 2.4.1 [The SFrame FRE Info Word], page 10.

It is important to underline here that although the canonical interpretation of these bytes is as stack offsets (to recover CFA, FP and RA), these bytes *may* be used by future ABIs/architectures to convey other information on a per SFrame FRE basis.

In summary, SFrame file format, by design, supports a variable number of stack offsets at the tail end of each SFrame FRE. To keep the SFrame file format specification flexible yet extensible, the interpretation of the stack offsets is ABI/arch-specific. The precise interpretation of the FRE stack offsets in the currently supported ABIs/architectures is covered in the ABI/arch-specific definition of the SFrame file format, See Chapter 3 [ABI/arch-specific Definition], page 12.

Next, the definitions of the three SFrame FRE types are as follows:

```
typedef struct sframe_frame_row_entry_addr1
{
  uint8_t sfre_start_address;
  sframe_fre_info sfre_info;
} ATTRIBUTE_PACKED sframe_frame_row_entry_addr1;

typedef struct sframe_frame_row_entry_addr2
{
  uint16_t sfre_start_address;
  sframe_fre_info sfre_info;
} ATTRIBUTE_PACKED sframe_frame_row_entry_addr2;

typedef struct sframe_frame_row_entry_addr4
{
  uint32_t sfre_start_address;
  sframe_fre_info sfre_info;
} ATTRIBUTE_PACKED sframe_frame_row_entry_addr4;
```

For ensuring compactness, SFrame frame row entries are stored unaligned on disk. Appropriate mechanisms need to be employed, as necessary, by the serializing and deserializing entities, if unaligned accesses need to be avoided.

`sfre_start_address` is an unsigned 8-bit/16-bit/32-bit integral field denoting the start address of a range of program counters, for which the SFrame FRE applies. The value encoded in the `sfre_start_address` field is the offset in bytes of the range's start address, from the start address of the function.

Further SFrame FRE types may be added in future.

2.4.1 The SFrame FRE Info Word

The SFrame FRE info word is a bitfield split into four parts. From MSB to LSB:

Bit offset	Name	Description
7	`fre_mangled_ra_p`	Indicate whether the return address is mangled with any authorization bits (signed RA).

5-6	`fre_offset_size`	Size of stack offsets in bytes. Valid values are: SFRAME_FRE_OFFSET_1B, SFRAME_FRE_OFFSET_2B, and SFRAME_FRE_OFFSET_4B.
1-4	`fre_offset_count`	A max value of 15 is allowed. Typically, a value of upto 3 is sufficient for most ABIs to track all three of CFA, FP and RA.
0	`fre_cfa_base_reg_id`	Distinguish between SP or FP based CFA recovery.

Name	**Value**	**Description**
`SFRAME_FRE_OFFSET_1B`	0	All stack offsets following the fixed-length FRE structure are 1 byte long.
`SFRAME_FRE_OFFSET_2B`	1	All stack offsets following the fixed-length FRE structure are 2 bytes long.
`SFRAME_FRE_OFFSET_4B`	2	All stack offsets following the fixed-length FRE structure are 4 bytes long.

3 ABI/arch-specific Definition

This section covers the ABI/arch-specific definition of the SFrame file format.

Currently, the only part of the SFrame file format definition that is ABI/arch-specific is the interpretation of the variable number of bytes at the tail end of each SFrame FRE. Currently, these bytes are used for representing stack offsets (for AMD64 and AARCH64 ABIs). For s390x ABI, the interpretation of these bytes may be stack offsets or even register numbers. It is recommended to peruse this section along with See Section 2.4 [SFrame Frame Row Entries], page 9, for clarity of context.

Future ABIs must specify the algorithm for identifying the appropriate SFrame FRE stack offsets in this chapter. This should inevitably include the blueprint for interpreting the variable number of bytes at the tail end of the SFrame FRE for the specific ABI/arch. Any further provisions, e.g., using the auxiliary SFrame header, etc., if used, must also be outlined here.

3.1 AMD64

Irrespective of the ABI, the first stack offset is always used to locate the CFA, by interpreting it as: CFA = `BASE_REG` + offset1. The identification of the `BASE_REG` is done by using the `fre_cfa_base_reg_id` field in the SFrame FRE info word.

In AMD64, the return address (RA) is always saved on stack when a function call is executed. Further, AMD64 ABI mandates that the RA be saved at a `fixed offset` from the CFA when entering a new function. This means that the RA does not need to be tracked per SFrame FRE. The fixed offset is encoded in the SFrame file format in the field `sfh_cfa_fixed_ra_offset` in the SFrame header. See Section 2.2 [SFrame Header], page 4.

Hence, the second stack offset (in the SFrame FRE), when present, will be used to locate the FP, by interpreting it as: FP = CFA + offset2.

Hence, in summary:

Offset ID	**Interpretation in AMD64**
1	CFA = `BASE_REG` + offset1
2	FP = CFA + offset2

3.2 AArch64

Irrespective of the ABI, the first stack offset is always used to locate the CFA, by interpreting it as: CFA = `BASE_REG` + offset1. The identification of the `BASE_REG` is done by using the `fre_cfa_base_reg_id` field in the SFrame FRE info word.

In AARCH64, the AAPCS64 standard specifies that the Frame Record saves both FP and LR (a.k.a the RA). However, the standard does not mandate the precise location in the function where the frame record is created, if at all. Hence the need to track RA in the SFrame stack trace format. As RA is being tracked in this ABI, the second stack offset is always used to locate the RA, by interpreting it as: RA = CFA + offset2. The third stack offset will be used to locate the FP, by interpreting it as: FP = CFA + offset3.

Given the nature of things, the number of stack offsets seen on AARCH64 per SFrame FRE is either 1 or 3.

Hence, in summary:

Offset ID	Interpretation in AArch64
1	CFA = `BASE_REG` + offset1
2	RA = CFA + offset2
3	FP = CFA + offset3

3.3 s390x

A stack tracer implementation must initialize the SP to the designated SP register value, the FP to the preferred FP register value, and the RA to the designated RA register value in the topmost stack frame of the callchain. This is required, as either the SP or FP is used as CFA base register and as the FP and/or RA are not necessarily saved on the stack. For RA this may only be the case in the topmost stack frame of the callchain. For FP this may be the case in any stack frame.

Irrespective of the ABI, the first stack offset is always used to locate the CFA. On s390x the value of the offset is stored adjusted by the s390x-specific `SFRAME_S390X_CFA_OFFSET_ADJUSTMENT` and scaled down by the s390x-specific `SFRAME_S390X_CFA_OFFSET_ALIGNMENT_FACTOR`, to enable and improve the use of signed 8-bit offsets on s390x. s390x-specific helpers `SFRAME_V2_S390X_CFA_OFFSET_ENCODE` and `SFRAME_V2_S390X_CFA_OFFSET_DECODE` are provided to perform or undo the adjustment and scaling. The CFA offset can therefore be interpreted as: CFA = `BASE_REG` + offset1 - `SFRAME_S390X_CFA_OFFSET_ADJUSTMENT` or CFA = `BASE_REG` + (offset1 * `SFRAME_S390X_CFA_OFFSET_ALIGNMENT_FACTOR`) - `SFRAME_S390X_CFA_OFFSET_ADJUSTMENT`. The identification of the `BASE_REG` is done by using the `fre_cfa_base_reg_id` field in the SFrame FRE info word.

The (64-bit) s390x ELF ABI does not mandate the precise location in a function where the return address (RA) and frame pointer (FP) are saved, if at all. Hence the need to track RA in the SFrame stack trace format. As RA is being tracked in this ABI, the second stack offset is always used to locate the RA stack slot, by interpreting it as: RA = CFA + offset2, unless the offset has a value of `SFRAME_FRE_RA_OFFSET_INVALID`. RA remains unchanged, if the offset is not available or has a value of `SFRAME_FRE_RA_OFFSET_INVALID`. Stack tracers are recommended to validate that the "unchanged RA" pattern, when present, is seen only for the topmost stack frame. The third stack offset is used to locate the FP stack slot, by interpreting it as: FP = CFA + offset3. FP remains unchanged, if the offset is not available.

In leaf functions the RA and FP may be saved in other registers, such as floating-point registers (FPRs), instead of on the stack. To represent this in the SFrame stack trace format the DWARF register number is encoded as RA/FP offset using the least-significant bit (LSB) as indication: offset = (regnum << 1) | 1. A LSB of zero indicates a stack slot offset. A LSB of one indicates a DWARF register number, which is interpreted as: regnum = offset >> 1. Given the nature of leaf functions, this can only occur in the topmost frame during stack tracing. It is recommended that a stack tracer implementation performs the required checks to ensure that restoring FP and RA from the said register locations is done only for topmost stack frame in the callchain.

Given the nature of things, the number of stack offsets and/or register numbers seen on s390x per SFrame FRE is either 1, 2, or 3.

Hence, in summary:

Offset ID	**Interpretation in s390x**
1	CFA = `BASE_REG` + offset1
2	RA stack slot = CFA + offset2, if (offset2 & 1 == 0) RA register number = offset2 >> 1, if (offset2 & 1 == 1) RA not saved if (offset2 == `SFRAME_FRE_RA_OFFSET_INVALID`)
3	FP stack slot = CFA + offset3, if (offset3 & 1 == 0) FP register number = offset3 >> 1, if (offset3 & 1 == 1)

The s390x ELF ABI defines the CFA as stack pointer (SP) at call site +160. The SP can therefore be obtained using the SP value offset from CFA `SFRAME_S390X_SP_VAL_OFFSET` of -160 as follows: SP = CFA + `SFRAME_S390X_SP_VAL_OFFSET`

Appendix A Generating Stack Traces using SFrame

Using some C-like pseudocode, this section highlights how SFrame provides a simple, fast and low-overhead mechanism to generate stack traces. Needless to say that for generating accurate and useful stack traces, several other aspects will need attention: finding and decoding bits of SFrame section(s) in the program binary, symbolization of addresses, to name a few.

In the current context, a `frame` is the abstract construct that encapsulates the following information:

- program counter (PC),
- stack pointer (SP), and
- frame pointer (FP)

With that said, establishing the first `frame` should be trivial:

```
// frame 0
frame->pc = current_IP;
frame->sp = get_reg_value (REG_SP);
frame->fp = get_reg_value (REG_FP);
```

where `REG_SP` and `REG_FP` are are ABI-designated stack pointer and frame pointer registers respectively.

Next, given frame N, generating stack trace needs us to get frame N+1. This can be done as follows:

```
// Get the PC, SP, and FP for frame N.
pc = frame->pc;
sp = frame->sp;
fp = frame->fp;
// Populate frame N+1.
int err = get_next_frame (&next_frame, pc, sp, fp);
```

where given the values of the program counter, stack pointer and frame pointer from frame N, `get_next_frame` populates the provided `next_frame` object and returns the error code, if any. In the following pseudocode for `get_next_frame`, the `sframe_*` functions fetch information from the SFrame section.

```
fre = sframe_find_fre (pc);
if (fre)
    // Whether the base register for CFA tracking is REG_FP.
    base_reg_val = sframe_fre_base_reg_fp_p (fre) ? fp : sp;
    // Get the CFA stack offset from the FRE.
    cfa_offset = sframe_fre_get_cfa_offset (fre);
    // Get the fixed RA offset or FRE stack offset as applicable.
    ra_offset = sframe_fre_get_ra_offset (fre);
    // Get the fixed FP offset or FRE stack offset as applicable.
    fp_offset = sframe_fre_get_fp_offset (fre);

    cfa = base_reg_val + cfa_offset;
    next_frame->sp = cfa [+ SFRAME_S390X_SP_VAL_OFFSET on s390x];
```

```
    ra_stack_loc = cfa + ra_offset;
    // Get the address stored in the stack location.
    next_frame->pc = read_value (ra_stack_loc);

    if (fp_offset is VALID)
        fp_stack_loc = cfa + fp_offset;
        // Get the value stored in the stack location.
        next_frame->fp = read_value (fp_stack_loc);
    else
        // Continue to use the value of fp as it has not
        // been clobbered by the current frame yet.
        next_frame->fp = fp;
else
    ret = ERR_NO_SFRAME_FRE;
```

Index

A

ABI/arch-specific Definition 12

C

Changes from Version 1 to Version 2 1

E

endianness 3

I

Introduction 1

O

Overview 1

P

Provisions for future ABIs 5, 7, 10

S

SFrame ABI/arch Identifier 6
SFrame FDE 6
SFrame Flags 4
SFrame FRE 9
SFrame header 4
SFrame magic number 3
SFrame preamble 3
SFrame Section 3
SFrame versions 3

T

The SFrame FDE Info Word 8
The SFrame FRE Info Word 10