The PDB Info Stream (aka the PDB Stream)

Stream Header

At offset 0 of the PDB Stream is a header with the following layout:

```
struct PdbStreamHeader {
  ulittle32_t Version;
  ulittle32_t Signature;
  ulittle32_t Age;
  Guid UniqueId;
};
```

- **Version** - A Value from the following enum:

```
enum class PdbStreamVersion : uint32_t {
  VC2 = 19941610,
  VC4 = 19950623,
  VC41 = 19950814,
  VC50 = 19960307,
  VC98 = 19970604,
  VC70Dep = 19990604,
  VC70 = 20000404,
  VC80 = 20030901,
  VC110 = 20091201,
  VC140 = 20140508,
};
```

While the meaning of this field appears to be obvious, in practice we have never observed a value other than `VC70`, even with modern versions of the toolchain, and it is unclear why the other values exist. It is assumed that certain aspects of the PDB stream's layout, and perhaps even that of the other streams, will change if the value is something other than `VC70`.

- **Signature** - A 32-bit time-stamp generated with a call to `time()` at the time the PDB file is written. Note that due to the inherent uniqueness problems of using a timestamp with 1-second granularity, this field does not really serve its intended purpose, and as such is typically ignored in favor of the `Guid` field, described below.
- **Age** - The number of times the PDB file has been written. This can be used along with `Guid` to match the PDB to its corresponding executable.
- **Guid** - A 128-bit identifier guaranteed to be unique across space and time. In general, this can be thought of as the result of calling the Win32 API UuidCreate, although LLVM cannot rely on that, as it must work on non-Windows platforms.

Named Stream Map

Following the header is a serialized hash table whose key type is a string, and whose value type is an integer. The existence of a mapping `X -> Y` means that the stream with the name `X` has stream index `Y` in the underlying MSF file. Note that not all streams are named (for example, the TPI Stream has a fixed index and as such there is no need to look up its index by name). In practice, there are usually

only a small number of named streams and these are enumerated in the table of streams in The PDB File Format. A corollary of this is if a stream does have a name (and as such is in the named stream map) then consulting the Named Stream Map is likely to be the only way to discover the stream's MSF stream index. Several important streams (such as the global string table, which is called `/names`) can only be located this way, and so it is important to both produce and consume this correctly as tools will not function correctly without it.

Important

Some streams are located by fixed indices (e.g TPI Stream has index 2), but other streams are located by fixed names (e.g. the string table is called `/names`) and can only be located by consulting the Named Stream Map.

The on-disk layout of the Named Stream Map consists of 2 components. The first is a buffer of string data prefixed by a 32-bit length. The second is a serialized hash table whose key and value types are both `uint32_t`. The key is the offset of a null-terminated string in the string data buffer specifying the name of the stream, and the value is the MSF stream index of the stream with said name. Note that although the key is an integer, the hash function used to find the right bucket hashes the string at the corresponding offset in the string data buffer.

The on-disk layout of the serialized hash table is described at The PDB Serialized Hash Table Format.

Note that the entire Named Stream Map is not length-prefixed, so the only way to get to the data following it is to de-serialize it in its entirety.

PDB Feature Codes

Following the Named Stream Map, and consuming all remaining bytes of the PDB Stream is a list of values from the following enumeration:

```
enum class PdbRaw_FeatureSig : uint32_t {
  VC110 = 20091201,
  VC140 = 20140508,
  NoTypeMerge = 0x4D544F4E,
  MinimalDebugInfo = 0x494E494D,
};
```

The meaning of these values is summarized by the following table:

Flag	Meaning
VC110	• No other features flags are present • PDB contains an IPI Stream
VC140	• Other feature flags may be present • PDB contains an IPI Stream
NoTypeMerge	• Presumably duplicate types can appear in the TPI Stream, although it's unclear why this might happen.
MinimalDebugInfo	• Program was linked with /DEBUG:FASTLINK • There is no TPI / IPI stream, all type info is contained in the original object files.

Matching a PDB to its executable

The linker is responsible for writing both the PDB and the final executable, and as a result is the only entity capable of writing the information necessary to match the PDB to the executable.

In order to accomplish this, the linker generates a guid for the PDB (or re-uses the existing guid if it is linking incrementally) and increments the Age field.

The executable is a PE/COFF file, and part of a PE/COFF file is the presence of number of "directories".

For our purposes here, we are interested in the “debug directory”. The exact format of a debug directory is described by the IMAGE_DEBUG_DIRECTORY structure. For this particular case, the linker emits a debug directory of type `IMAGE_DEBUG_TYPE_CODEVIEW`. The format of this record is defined in `llvm/DebugInfo/CodeView/CVDebugRecord.h`, but it suffices to say here only that it includes the same `Guid` and `Age` fields. At runtime, a debugger or tool can scan the COFF executable image for the presence of a debug directory of the correct type and verify that the Guid and Age match.