Making Backwards-Compatible Changes in the LLVM Bitstream Format

Swift uses the LLVM bitstream format for some of its serialization logic. This format was invented as a container for LLVM IR. It is a binary format supporting two basic structures: *blocks,* which define regions of the file, and *records,* which contain data fields that can be up to 64 bits. It has a few nice properties that make it a useful container format for us as well:

- It is easy to skip over an entire block, because the block's length is recorded at its start.
- It is possible to jump to specific offsets *within* a block without having to reparse from the start of the block.
- A format change doesn't immediately invalidate existing bitstream files, because the stream includes layout information for each record.

However, it has some disadvantages as well:

- Each record can only contain one variable-sized entry (either an array or a "blob" of bytes).
- Higher-level features like cross-references or lookup by key have to be built on top of the format, usually in a way that the existing tooling doesn't understand.

You can view the contents of any LLVM bitstream using the `llvm-bcanalyzer` tool's `-dump` option.

Backwards-compatibility

For a format change to be backwards-compatible, we need the v5 tools to be able to read a file generated by the v6 tools. At a high level, this means that whatever data is introduced in v6, it doesn't interfere with what v5 is looking for.

(We also care about *forwards*-compatibility, which says that the v6 tools is able to read a file generated by the v5 tools. This is usually easier to maintain, because the v5 format is already known.)

In practice, there are a few ways to accomplish this with LLVM bitstreams:

- If the deserialization logic is set to skip over any blocks it doesn't understand, a new format can always add new blocks.
- If the deserialization logic is set to skip over any *records* it doesn't understand, a new format can always add new *records*. Be careful, though, of records that are expected to appear immediately after another record---if you put a new record between them, you may break the expectations of older compilers.
- If the deserialization logic always looks for a possible blob entry in records (i.e. passing a StringRef out-parameter to BitstreamCursor's `readRecord`), a new format can add blob data to an existing record that does not have it.
- If the deserialization logic always checks for a minimum number of fields in a record before extracting those fields, or if the only field in a record is blob data, a new format can add new fields to an existing record, as long as they come after any existing non-blob fields.

  Note that the BCRecordLayout DSL expects the number of fields to **match exactly**. If you want to use BCRecordLayout's `readRecord` method, the deserialization logic will have to check that the deserialized data has the correct number of fields ahead of time. If it has more fields, you can make an ArrayRef that

slices off the extra ones; if it has fewer, you're reading from an old format and will need to use a different BCRecordLayout, or just read them manually.

(We could also add more API to BCRecordLayout to make this easier. It's part of LLVM, but it's a part of LLVM originally contributed by Swift folks.)

Note also that it's still okay to use BCRecordLayout for *serialization*. It's only deserialization where we have to be careful about multiple formats.

Remember that any new data will be *ignored* by the old tools. If it's something that *should* affect how old tools read the file, it must be encoded in an existing field; if that's impossible, you have a backwards-incompatible change and should bump the major version number of the file.

If the existing deserialization logic is already checking for the exact size of a record (and therefore preventing new fields from being added), one trick is to put a second record after the first, and check for its presence in the new version of the tools. As long as the old logic is set up to skip unknown records, this shouldn't cause any problems.