llvm::Error

Rich Error Handling in LLVM

Error Handling History

- LLVM’s APIs historically used ad-hoc approaches
  - bools, nullptrs, string errors…
- std::error_code
  - C++ standard library error type
  - Enumerable errors only
- Lack of enforcement — easy to drop errors

Exceptions

- User defined error types
- Type safe handlers
- Once thrown, impossible to forget

However…

- Not actually zero-cost
- Turned off in LLVM

llvm::Error

- Error as return value:

```
    Error foo(…);
    Expected<T> bar(…);
```

- User defined error types:

```
class MyError
   : public ErrorInfo<MyError> { … };
```

- Errors must be checked before destruction

Idiomatic Usage

```
Error foo(…);

if (auto Err = foo(…))
  return Err;
```

Conversion to bool "checks" error

Idiomatic Usage

```
Error foo(…);

foo(…);
```

Destruction of unchecked
Error triggers abort

Type-safe Handlers

```
Error foo(…);

handleErrors(
   foo(…),
   [](MyError &M) {
   },
   [](SomeOtherError &S) {
   },
   …,
);
```

Benefits

- Safer: Avoid vulnerabilities due to missed errors
- More descriptive:
  - `LLVM ERROR: Malformed MachO file.`

  becomes

  -
    ```
    truncated or malformed object
     (bad section index: 66 for symbol at index 8)
    ```
- Supports error hierarchies (e.g. ObjectFileError)

Conclusion

- Many utilities:
  - Interoperability with std::error_code and ErrorOr
  - Standard error types (StringError)
  - Exit-on-error idiom support for tool code
- Check out Programmers Manual for usage