

# The Data Inspection Language: Fast Simple Expression Evaluation in LLDB

Ilya Kuklin (Access Softek)

Caroline Tice (Google)

Anton Korobeynikov (Access Softek)

Andrei Lebedev (Access Softek)

# Motivation

- Debugging is frustrating when every step can take a noticeable time to complete
  - LLDB spends a substantial amount of time evaluating expressions needed to display debug summaries in an IDE
  - It is a common problem in very large projects
- Debugging is fragile and unstable
  - Single **print x** could crash a debugger if it stopped in some weird context
  - Expression evaluation relies on Clang AST which has a lot of internal invariants and is tricky to construct

# Example

- `FName` represents a string, but only stores an index to a global table of strings

```
class FName {  
    // ...  
    FNameEntryId ComparisonIndex;  
    // ...  
};
```

# Part of the Python formatters, simplified:

```
def UEFNameIndexToEntry(EntryId):  
    Index = EntryId.GetChildMemberWithName('Value').GetValueAsUnsigned(0)  
    NameEntryExpr = f'(FNameEntry*)(GNameBlocksDebug[{Index} >> 16] + (2 * ({Index} & 16)))'  
    NameEntry = EntryId.CreateValueFromExpression('NameEntry', NameEntryExpr)  
    return NameEntry
```

VS Code:

```
▼ EventName = 'name=b'ExtensionAdded''  
▼ ComparisonIndex = FNameEntryId @ 0x7fffffffcc358  
    Value = 296599
```

- These debugging expressions tend to be small and simple
- They can be evaluated much faster by an interpreter, rather than a full compiler

# Example

- `FName` represents a string, but only stores an index to a global table of strings

```
class FName {
// ...
FNameEntryId ComparisonIndex;
// ...
};

# Part of the Python formatters, simplified:
def UEFNameIndexToEntry(EntryId):
    Index = EntryId.GetChildMemberWithName('Value').GetValueAsUnsigned(0)
    NameEntryExpr = f'(FNameEntry*)(GNameBlocksDebug[{Index} >> 16] + (2 * ({Index} & 16)))'
    NameEntry = EntryId.CreateValueFromExpression('NameEntry', NameEntryExpr)
    return NameEntry
```

VS Code:

```
▼ EventName = 'name=b'ExtensionAdded''
▼ ComparisonIndex = FNameEntryId @ 0x7fffffffcc358
    Value = 296599
```

- These debugging expressions tend to be small and simple
- They can be evaluated much faster by an interpreter, rather than a full compiler

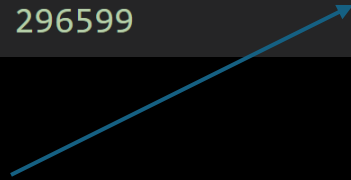
# Example

- `FName` represents a string, but only stores an index to a global table of strings

```
class FName {  
    // ...  
    FNameEntryId ComparisonIndex;  
    // ...  
};  
  
# Part of the Python formatters, simplified:  
def UEFNameIndexToEntry(EntryId):  
    Index = EntryId.GetChildMemberWithName('Value').GetValueAsUnsigned(0)  
    NameEntryExpr = f'(FNameEntry*)(GNameBlocksDebug[{Index} >> 16] + (2 * ({Index} & 16)))'  
    NameEntry = EntryId.CreateValueFromExpression('NameEntry', NameEntryExpr)  
    return NameEntry
```

VS Code:

```
▼ EventName = 'name=b'ExtensionAdded'  
▼ ComparisonIndex = FNameEntryId @ 0x7fffffffcc358  
    Value = 296599
```



- These debugging expressions tend to be small and simple
- They can be evaluated much faster by an interpreter, rather than a full compiler

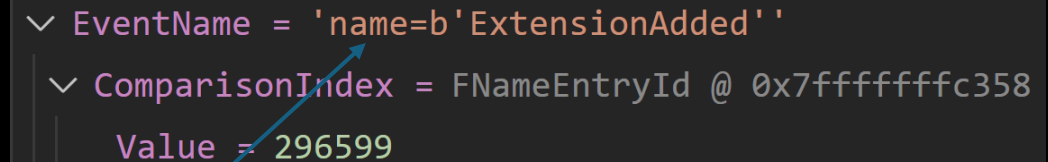
# Example

- `FName` represents a string, but only stores an index to a global table of strings

```
class FName {
// ...
FNameEntryId ComparisonIndex;
// ...
};

# Part of the Python formatters, simplified:
def UEFNameIndexToEntry(EntryId):
    Index = EntryId.GetChildMemberWithName('Value').GetValueAsUnsigned(0)
    NameEntryExpr = f'(FNameEntry*)(GNameBlocksDebug[{Index} >> 16] + (2 * ({Index} & 16)))'
    NameEntry = EntryId.CreateValueFromExpression('NameEntry', NameEntryExpr)
    return NameEntry
```

VS Code:



```
▼ EventName = 'name=b'ExtensionAdded''
▼ ComparisonIndex = FNameEntryId @ 0x7fffffffcc358
    Value = 296599
```

- These debugging expressions tend to be small and simple
- They can be evaluated much faster by an interpreter, rather than a full compiler

# History

- lldb-eval project was implemented as a fast interpreter for a subset of C++
  - Andy Yankovsky. Building a faster expression evaluator for LLDB – LLVM Developers’ Meeting 2021.
- We revamped the project, rebased it on the mainline, and experimented with it as the first phase of expression evaluation with a fallback to full LLDB evaluation
  - Ilya Kuklin et al. Experiments with two-phase expression evaluation for a better debugging experience – LLVM Developers' Meeting 2024.

|                                          | lldb-eval (via LLDB) | LLDB    | LLDB + lldb-eval overhead |
|------------------------------------------|----------------------|---------|---------------------------|
| 1 expression                             | 0.65 ms              | 87.9 ms | 88.9 ms                   |
| Total for all local and global variables | 103 ms               | 2025 ms | 2041 ms                   |

# The Data Inspection Language (DIL)

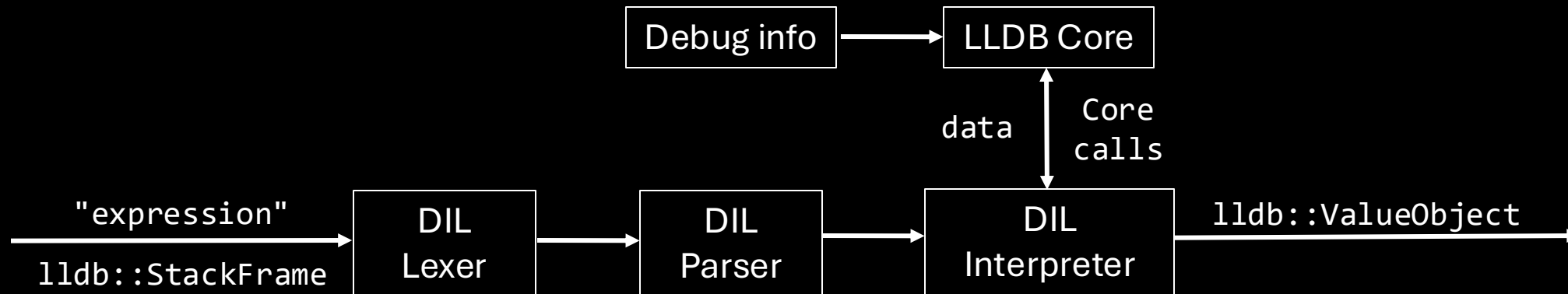


- Expression language designed to inspect the data in the program
- Designed to be a core part of LLDB for better integration, to reduce overhead and code redundancy
- Allows programming language-agnostic inspection of data in the program
- Supports arithmetic, logic, comparison, data retrieval operations, type casts
- Intended to replace the existing ``frame variable`` command implementation
- RFC for DIL by Andy Yankovsky: <https://discourse.llvm.org/t/rfc-data-inspection-language/69893>



# DIL overall design

- Evaluates expressions using its own lexer, parser and interpreter and relies purely on the debug information retrieved from LLDB
- A newly written lexer, independent from any frontend
- Most of the parser/interpreter logic is inspired by lldb-eval



# Current status

- Landing feature by feature to upstream LLDB
  - Heavier use of LLDB core functions
  - Additional changes to other LLDB code itself making it faster
- For now, the syntax is very similar to a small subset of C++,
  - Can be expanded to support the syntax of multiple programming languages when needed
- Already in the mainline: dereference, address of, subscripts and field retrieval
- Partially based on lldb-eval
- ``frame variable`` implementation uses DIL engine under the hood

# Work in progress



- Could be turn on/off using the setting 'target.experimental.use-DIL <true/false>'
- First milestone is to replace the old ``frame variable``
- More DIL features to support simple expressions, i.e. type casts and all arithmetic and logic operations
- Add basic function calls
- Prove programming language agnostic approach by supporting features for languages like Swift

# Preliminary DIL performance

- Result for work in progress DIL

## DIL debugging performance

|                                             | DIL (via LLDB) | LLDB    |
|---------------------------------------------|----------------|---------|
| 1 expression                                | 14 ms          | 67 ms   |
| Total for all<br>local and global variables | 287 ms         | 1353 ms |

# Conclusion

- DIL can already be used to evaluate simple expressions either via ``frame var`` command, API call, or direct LLDB function call
  - `'target.experimental.use-DIL <true/false>'` setting to change between DIL and the original ``frame var`` implementation
- Can be used via ``frame var`` and LLDB-DAP out of the box
- Expanding DIL capabilities automatically improves LLDB performance as well as debugging speed via LLDB-DAP

D F X G E W J Q  
R A L N P K Y H  
Z B C C Y V R U  
E N J W E Q M S  
O T B X G S P  
F H K T D V Z M

ACCESS SOFTEK, INC

# Thank you!