

Minimal Structured Logging for Autonomous Vehicles (AVs)

Date: 09/18/2019

Prepared by: Robert Keelan
Email: rkeelan@argo.ai

Agenda

- AV Debug Log Users and Requirements
- High Level Picture of “Ideal” Debug Log System
- Debug Log System Implementation in C++17
 - C++20 Improvements
- Error Handling Integration



Debug Logs and Autonomous Vehicles

- Debug Log
 - A record of events that occurs during the execution of a program
- Useful tool to help determine what has happened on a vehicle.
 - Vehicle Operators
 - Triage Engineers
 - Developers
 - Cloud Tooling

Sample Log Statement Content	
Field	Sample Value
Filename	Task.cc
Function Name	foo()
Line Number	107
Semantic Level	ERROR
Timestamp	5678.987
Payload	Tried to access index 7 on vector of size 6

Logging Paradigms

Logging Paradigms	
Text Logging	Structured Logging
"5678.987 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6"	{ "filename": "Task.cc", "function": "foo()", "line": 107, "level": "ERROR", "timestamp": 5678.987 "payload": "Tried to access index 7 on vector of size 6" }

Logging Paradigms

Logging Paradigms

Text Logging

“5678.987 [ERROR] Task.cc:107 foo()
Tried to access index 7 on vector of
size 6”

```
5678.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5679.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5680.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5681.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5682.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5683.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5684.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5685.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5686.987 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6  
5687.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5688.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad
```

Structured Logging

```
{  
  "filename": "Task.cc",  
  "function": "foo()",  
  "line": 107,  
  "severity": "ERROR",  
  "timestamp": 5678.987  
  "message": "Tried to access index 7 on  
vector of size 6"
```

Logging Paradigms

Logging Paradigms

Text Logging

“5678.987 [ERROR] Task.cc:107 foo()
Tried to access index 7 on vector of
size 6”

```
5678.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5679.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5680.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5681.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5682.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5683.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5684.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5685.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5686.987 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6  
5687.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad  
5688.987 [INFO] Task.cc:100 baz() Lorem ipsum dolor sit amet, consectetur ad
```


Structured Logging

```
{  
  "filename": "Task.cc",  
  "function": "foo()",  
  "line": 107,  
  "severity": "ERROR",  
  "timestamp": 5678.987  
  "message": "Tried to access index 7 on  
vector of size 6"
```

Logging Paradigms

Logging Paradigms	
Text Logging	Structured Logging
"5678.987 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6"	{ "filename": "Task.cc", "function": "foo()", "line": 107, "level": "ERROR", "timestamp": 5678.987 "payload": "Tried to access index 7 on vector of size 6" }

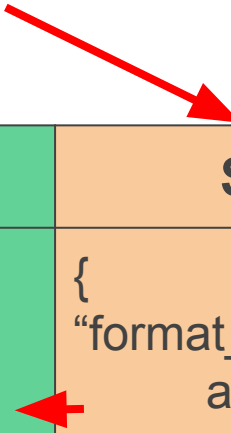
Logging Paradigms

Logging Paradigms	
Text Logging	Structured Logging
<p>“5678.987 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6”</p> <div><p>“payload”: { “string”: “Invalid vector access”, “index: 7, “size: 6 }</p></div>	<pre>{ “filename”: “Task.cc”, “function”: “foo()”, “line”: 107, “level”: “ERROR”, “timestamp”: 5678.987 “payload”: “Tried to access index 7 on vector of size 6” }</pre> 

“Ideal” Debug Log Interface

```
LOG_ERROR(“Tried to access index {{index}} on vector of size  
{{size}}.”, idx, siz);
```

- Macro for each debug log level
 - **macro vs. function**
- *std::format* / *{fmt}* compatible format string (mostly)
- **Still positional fields**



Text	Structured
“Tried to access element 7 on container of size 6.”	{ “format_string”: “Tried to access index {} on vector of size {}.” “index”: 7, “size”: 6 }

Metadata vs Instance Data

```
5678.987 [ERROR] Task.cc:107 foo() Tried to access index 1 on vector of size 0
5679.876 [ERROR] Task.cc:107 foo() Tried to access index 2 on vector of size 1
5680.765 [ERROR] Task.cc:107 foo() Tried to access index 3 on vector of size 2
5681.654 [ERROR] Task.cc:107 foo() Tried to access index 4 on vector of size 3
5682.543 [ERROR] Task.cc:107 foo() Tried to access index 5 on vector of size 4
5683.432 [ERROR] Task.cc:107 foo() Tried to access index 6 on vector of size 5
5684.321 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6
5685.219 [ERROR] Task.cc:107 foo() Tried to access index 8 on vector of size 7
5686.198 [ERROR] Task.cc:107 foo() Tried to access index 9 on vector of size 8
5687.987 [ERROR] Task.cc:107 foo() Tried to access index 8 on vector of size 7
5688.876 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6
5689.765 [ERROR] Task.cc:107 foo() Tried to access index 6 on vector of size 5
```

Metadata vs Instance Data

```
5678.987 [ERROR] Task.cc:107 foo() Tried to access index 1 on vector of size 0
5679.876 [ERROR] Task.cc:107 foo() Tried to access index 2 on vector of size 1
5680.765 [ERROR] Task.cc:107 foo() Tried to access index 3 on vector of size 2
5681.654 [ERROR] Task.cc:107 foo() Tried to access index 4 on vector of size 3
5682.543 [ERROR] Task.cc:107 foo() Tried to access index 5 on vector of size 4
5683.432 [ERROR] Task.cc:107 foo() Tried to access index 6 on vector of size 5
5684.321 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6
5685.219 [ERROR] Task.cc:107 foo() Tried to access index 8 on vector of size 7
5686.198 [ERROR] Task.cc:107 foo() Tried to access index 9 on vector of size 8
5687.987 [ERROR] Task.cc:107 foo() Tried to access index 8 on vector of size 7
5688.876 [ERROR] Task.cc:107 foo() Tried to access index 7 on vector of size 6
5689.765 [ERROR] Task.cc:107 foo() Tried to access index 6 on vector of size 5
```

Metadata

Instance Data

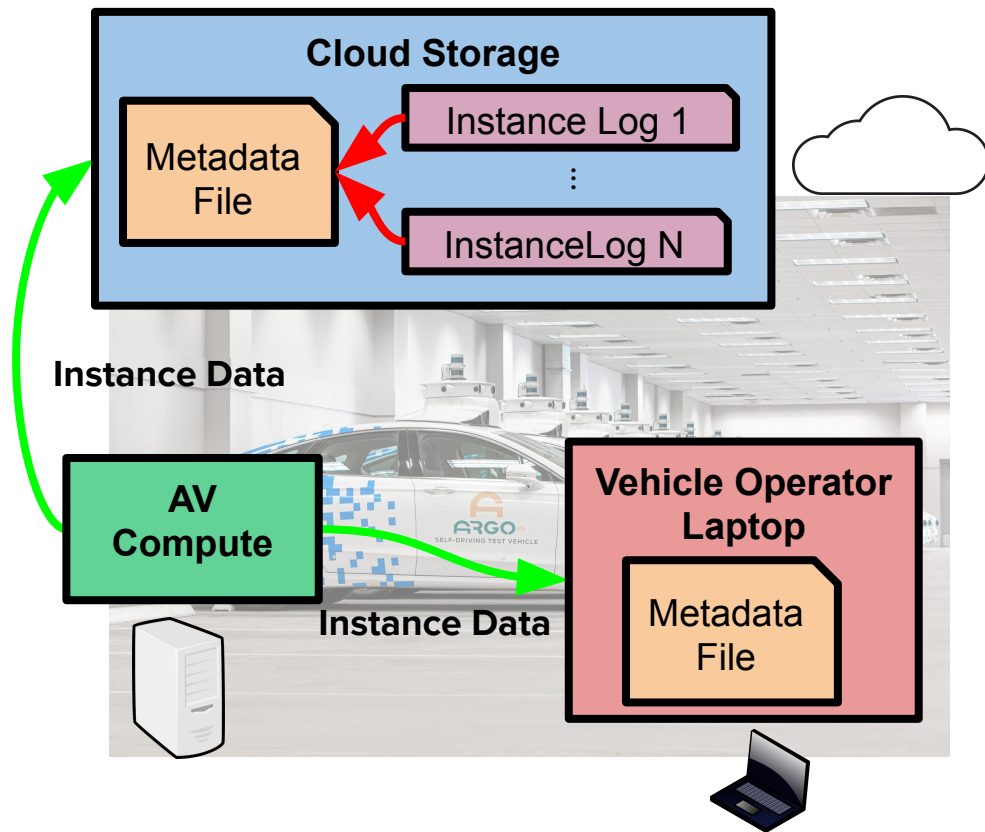
“Ideal” AV Debug Logger

- Structured metadata
 - Logged once (per version)
- Instance Data
 - Metadata ID
 - Timestamp
 - Dynamic payload data
- Serialized instance data logged on vehicle
- Text based logs
 - Generated on the fly from instance and metadata



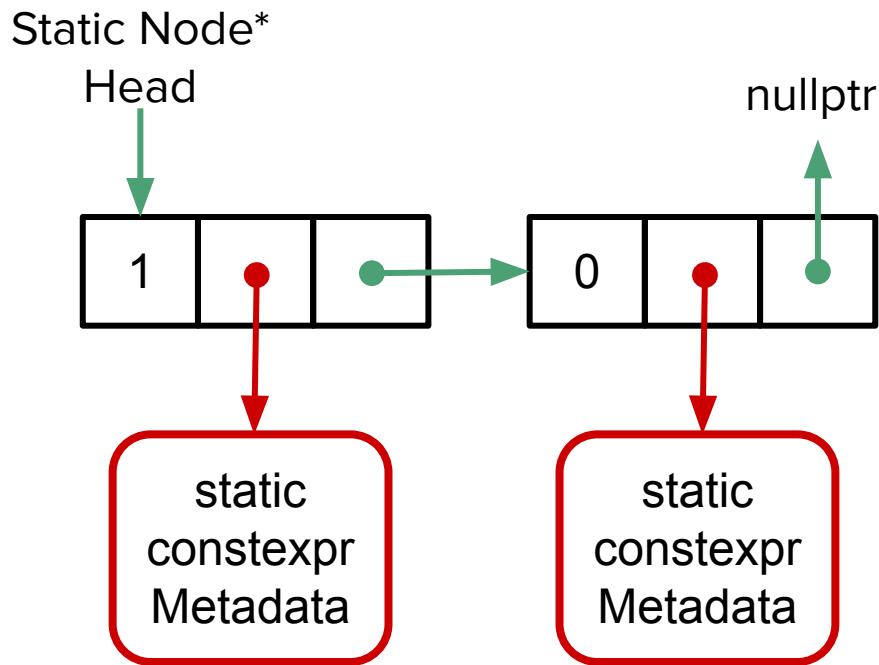
“Ideal” AV Debug Logger

- Structured metadata
 - Logged once (per version)
- Instance Data
 - Metadata ID
 - Timestamp
 - Dynamic payload data
- Serialized instance data logged on vehicle
- Text based logs
 - Generated on the fly from instance and metadata



Metadata Aggregation

- Statically generate an intrusive singly linked list
- Unique ID assigned to every list element.
- Metadata written to a file.



Debug Log Statement Contents

```
enum class Level;
```

```
struct LogMacroData {  
    std::string_view file{};  
    std::string_view function{};  
    std::string_view fmt_str{};  
    int32_t line{};  
    Level level{};  
};
```

```
struct LogStatementMetadata {  
    LogMacroData macro_data{};  
    TypeDescriptors descriptors{};  
};
```

Debug Log Statement Contents

```
enum class Level;
```

```
struct LogMacroData {  
    std::string_view file{};  
    std::string_view function{};  
    std::string_view fmt_str{};  
    int32_t line{};  
    Level level{};  
};
```

```
struct LogStatementMetadata {  
    LogMacroData macro_data{};  
    TypeDescriptors descriptors{};  
};
```


Metadata Aggregation: Log Macro

```
#define LOG_ERROR(format, ...) \
do { \
    static constexpr std::string_view anonymous_function{__FUNCTION__}; \
    struct { \
        constexpr LogMacroData operator()() const noexcept { \
            return LogMacroData{format, __FILE__, anonymous_function, \
                                __LINE__, Level::ERROR}; \
        } \
    } anonymous_meta_data; \
    log<decltype(anonymous_meta_data)>(__VA_ARGS__); \
} while(false);
```

```
void foo() { LOG_ERROR("Tried to access index {{index}} on vector of  
size {{size}}.", idx, siz); }
```

Metadata Aggregation: Log Macro

```
#define LOG_ERROR(format, ...) \
do { \
    static constexpr std::string_view anonymous_function{__FUNCTION__}; \
    struct { \
        constexpr LogMacroData operator()() const noexcept { \
            return LogMacroData{format, __FILE__, anonymous_function, \
                                __LINE__, Level::ERROR}; \
        } \
    } anonymous_meta_data; \
    log<decltype(anonymous_meta_data)>(__VA_ARGS__); \
} while(false);
```



```
void foo() { LOG_ERROR("Tried to access index {{index}} on vector of \
size {{size}}.", idx, siz); }
```

Metadata Aggregation: Log Macro

```
#define LOG_ERROR(format, ...) \
do { \
    static constexpr std::string_view anonymous_function{__FUNCTION__}; \
    struct { \
        constexpr LogMacroData operator()() const noexcept { \
            return LogMacroData{format, __FILE__, anonymous_function, \
                                __LINE__, Level::ERROR}; \
        } \
    } anonymous_meta_data; \
    log<decltype(anonymous_meta_data)>(__VA_ARGS__); \
} while(false); \

void foo() { LOG_ERROR("Tried to access index {{index}} on vector of \
    size {{size}}.", idx, siz); }
```

Metadata Aggregation: Log Macro

```
#define LOG_ERROR(format, ...) \
do { \
    static constexpr std::string_view anonymous_function{__FUNCTION__}; \
    struct { \
        constexpr LogMacroData operator()() const noexcept { \
            return LogMacroData{format, __FILE__, anonymous_function, \
                                __LINE__, Level::ERROR}; \
        } \
    } anonymous_meta_data; \
    log<decltype(anonymous_meta_data)>(__VA_ARGS__); \
} while(false); \

void foo() { LOG_ERROR("Tried to access index {{index}} on vector of \
    size {{size}}.", idx, siz); }
```

Metadata Aggregation: Log Macro

```
#define LOG_ERROR(format, ...) \
do { \
    static constexpr std::string_view anonymous_function{__FUNCTION__}; \
    struct { \
        constexpr LogMacroData operator()() const noexcept { \
            return LogMacroData{format, __FILE__, anonymous_function, \
                                __LINE__, Level::ERROR}; \
        } \
    } anonymous_meta_data; \
    log<decltype(anonymous_meta_data)>(__VA_ARGS__); \
} while(false); \

void foo() { LOG_ERROR("Tried to access index {{index}} on vector of \
    size {{size}}.", idx, siz); }
```

Metadata Aggregation: Log Macro

```
#define LOG_ERROR(format, ...) \
do { \
    static constexpr std::string_view anonymous_function{__FUNCTION__}; \
    struct { \
        constexpr LogMacroData operator()() const noexcept { \
            return LogMacroData{format, __FILE__, anonymous_function, \
                                __LINE__, Level::ERROR}; \
        } \
    } anonymous_meta_data; \
    log<decltype(anonymous_meta_data)>(__VA_ARGS__); \
} while(false);
```

```
void foo() { LOG_ERROR("Tried to access index {{index}} on vector of \
size {{size}}.", idx, siz); }
```

Metadata Aggregation: Log Macro C++20

```
template<unsigned N>
struct FixedString {
    constexpr FixedString(char const* s);
    char buf[N + 1]{};
};

template<unsigned N>
FixedString(char const (&)[N]) -> FixedString<N - 1>;

template<FixedString fixed_string>
void log();

void foo() { log<__FUNCTION__>(); }
```

Metadata Aggregation: Log Macro C++20

```
template<unsigned N>
struct FixedString {
    constexpr FixedString(char const* s);
    char buf[N + 1]{};
};
template<unsigned N>
FixedString(char const (&)[N]) -> FixedString<N - 1>;
```

```
template<FixedString fixed_string>
void log();
```

```
void foo() { log<__FUNCTION__>(); }
```


Metadata Aggregation: Log Macro C++20

```
template<unsigned N>
struct FixedString {
    constexpr FixedString(char const* s);
    char buf[N + 1]{};
};
template<unsigned N>
FixedString(char const (&)[N]) -> FixedString<N - 1>;
```

```
template<FixedString fixed_string>
void log();
```

```
void foo() { log<__FUNCTION__>(); }
```

Metadata Aggregation: Linked List

```
int32_t gen_id() {
    static int32_t id{-1};
    ++id;
    return id;
}

struct LogDataNode;

LogDataNode*& log_data_head() {
    static LogDataNode* head{nullptr};
    return head;
}
```

```
struct LogDataNode {
    LogDataNode(DebugLogMetaData const* d) :
        id{gen_id()}, data{d}
    {
        auto& head = log_data_head();
        next = std::exchange(head, this);
    }

    int32_t id{};
    LogDataNode const* next;
    DebugLogMetaData const* data;
};
```

Metadata Aggregation: Linked List

```
int32_t gen_id() {  
    static int32_t id{-1};  
    ++id;  
    return id;  
}  
  
struct LogDataNode;  
  
LogDataNode*& log_data_head() {  
    static LogDataNode* head{nullptr};  
    return head;  
}
```

```
struct LogDataNode {  
    LogDataNode(DebugLogMetaData const* d) :  
        id{gen_id()}, data{d}  
    {  
        auto& head = log_data_head();  
        next = std::exchange(head, this);  
    }  
  
    int32_t id{};  
    LogDataNode const* next;  
    DebugLogMetaData const* data;  
};
```

Metadata Aggregation: Linked List

```
int32_t gen_id() {
    static int32_t id{-1};
    ++id;
    return id;
}

struct LogDataNode;

LogDataNode*& log_data_head() {
    static LogDataNode* head{nullptr};
    return head;
}
```

```
struct LogDataNode {
    LogDataNode(DebugLogMetaData const* d) :
        id{gen_id()}, data{d}
    {
        auto& head = log_data_head();
        next = std::exchange(head, this);
    }

    int32_t id{};
    LogDataNode const* next;
    DebugLogMetaData const* data;
};
```


Metadata Aggregation: Linked List

```
int32_t gen_id() {  
    static int32_t id{-1};  
    ++id;  
    return id;  
}
```

```
struct LogDataNode;
```

```
LogDataNode*& log_data_head() {  
    static LogDataNode* head{nullptr};  
    return head;  
}
```

```
struct LogDataNode {  
    LogDataNode(DebugLogMetaData const* d) :  
        id{gen_id()}, data{d}  
    {  
        auto& head = log_data_head();  
        next = std::exchange(head, this);  
    }  
  
    int32_t id{};  
    LogDataNode const* next;  
    DebugLogMetaData const* data;  
};
```



Metadata Aggregation: Linked List

```
int32_t gen_id() {  
    static int32_t id{-1};  
    ++id;  
    return id;  
}
```

```
struct LogDataNode;
```

```
LogDataNode*& log_data_head() {  
    static LogDataNode* head{nullptr};  
    return head;  
}
```

```
struct LogDataNode {  
    LogDataNode(DebugLogMetaData const* d) :  
        id{gen_id()}, data{d}  
    {  
        auto& head = log_data_head();  
        next = std::exchange(head, this);  
    }  
  
    int32_t id{};  
    LogDataNode const* next;  
    DebugLogMetaData const* data;  
};
```

Metadata Aggregation: Linked List

```
int32_t gen_id() {  
    static int32_t id{-1};  
    ++id;  
    return id;  
}  
  
struct LogDataNode;  
  
LogDataNode*& log_data_head() {  
    static LogDataNode* head{nullptr};  
    return head;  
}
```

```
struct LogDataNode {  
    LogDataNode(DebugLogMetaData const* d) :  
        id{gen_id()}, data{d}  
    {  
        auto& head = log_data_head();  
        next = std::exchange(head, this);  
    }  
  
    int32_t id{};  
    LogDataNode const* next;  
    DebugLogMetaData const* data;  
};
```

Metadata Aggregation: Construction

```
template <class F, class... Args>  
LogStatementMetadata const* get_meta_data_ptr();
```

```
template <class F, class... Args>  
inline MetaDataNode meta_data_node{get_meta_data_ptr()};
```


```
template <class F, class ...Args>  
void log(Args const& ...args) {  
    serialize(meta_data_node<F, Args...>.id, args...);  
};
```


Metadata Aggregation: Construction

```
template <class F, class... Args>  
LogStatementMetadata const* get_meta_data_ptr();
```

```
template <class F, class... Args>  
inline MetaDataNode meta_data_node{get_meta_data_ptr()};
```

```
template <class F, class ...Args>  
void log(Args const& ...args) {  
    serialize(meta_data_node<F, Args...>.id, args...);  
};
```



Metadata Aggregation: Construction

```
template <class F, class... Args>
```

```
LogStatementMetadata const* get_meta_data_ptr();
```

```
template <class F, class... Args>
```

```
inline MetaDataNode meta_data_node{get_meta_data_ptr()};
```

```
template <class F, class ...Args>
```

```
void log(Args const& ...args) {
```

```
    serialize(meta_data_node<F, Args...>.id, args...);
```

```
};
```

Metadata Aggregation: Construction

```
template <class F, class... Args>  
LogStatementMetadata const* get_meta_data_ptr();
```

```
template <class F, class... Args>  
inline MetaDataNode meta_data_node{get_meta_data_ptr()};
```

```
template <class F, class ...Args>  
void log(Args const& ...args) {  
    serialize(meta_data_node<F, Args...>.id, args...);  
};
```

Metadata Aggregation: TypeDescriptor

```
struct Enum {  
    Span<std::pair<int, std::string_view>> table;  
};  
struct Int{};  
struct Float{};
```

```
using TypeDescriptor = std::variant<Int, Float, Enum/*,...*/>;  
using TypeDescriptors = Span<TypeDescriptor>;
```

```
template <class T>  
struct GetTypeDescriptor;
```

Metadata Aggregation: TypeDescriptor

```
struct Enum {  
    Span<std::pair<int, std::string_view>> table;  
};
```

```
struct Int{};  
struct Float{};
```

```
using TypeDescriptor = std::variant<Int, Float, Enum/*,...*/>;  
using TypeDescriptors = Span<TypeDescriptor>;
```

```
template <class T>  
struct GetTypeDescriptor;
```

Metadata Aggregation: TypeDescriptor

```
struct Enum {  
    Span<std::pair<int, std::string_view>> table;  
};
```

```
struct Int{};  
struct Float{};
```

```
using TypeDescriptor = std::variant<Int, Float, Enum/*,...*/>;  
using TypeDescriptors = Span<TypeDescriptor>;
```

```
template <class T>  
struct GetTypeDescriptor;
```

Metadata Aggregation: TypeDescriptor

```
struct Enum {  
    Span<std::pair<int, std::string_view>> table;  
};  
struct Int{};  
struct Float{};  
  
using TypeDescriptor = std::variant<Int, Float, Enum/*,...*/>;  
using TypeDescriptors = Span<TypeDescriptor>;  
  
template <class T>  
struct GetTypeDescriptor;
```

Metadata Aggregation: TypeDescriptor

```
struct Enum {  
    Span<std::pair<int, std::string_view>> table;  
};  
struct Int{};  
struct Float{};  
  
using TypeDescriptor = std::variant<Int, Float, Enum/*,...*/>;  
using TypeDescriptors = Span<TypeDescriptor>;
```

```
template <class T>  
struct GetTypeDescriptor;
```


Metadata Aggregation: Metadata Generation

```
template <>
struct GetTypeDescriptor<int> {
    static constexpr TypeDescriptor value{Int{}}};
};
```

```
template <class F, class... Args>
LogStatementMetadata const* get_meta_data_ptr() {
    static constexpr std::array
        type_descriptors{GetTypeDescriptor<Args>::value...};
    static constexpr LogStatementMetadata
        meta_data{F{}}(), Span{type_descriptors}};
    return &meta_data;
}
```

Metadata Aggregation: Metadata Generation

```
template <>
struct GetTypeDescriptor<int> {
    static constexpr TypeDescriptor value{Int{}}};
};
```

```
template <class F, class... Args>
LogStatementMetadata const* get_meta_data_ptr() {
    static constexpr std::array
        type_descriptors{GetTypeDescriptor<Args>::value...};
    static constexpr LogStatementMetadata
        meta_data{F{}}(), Span{type_descriptors}};
    return &meta_data;
}
```

Metadata Aggregation: Metadata Generation

```
template <>
struct GetTypeDescriptor<int> {
    static constexpr TypeDescriptor value{Int{}}};
};

template <class F, class... Args>
LogStatementMetadata const* get_meta_data_ptr() {
    static constexpr std::array
        type_descriptors{GetTypeDescriptor<Args>::value...};
    static constexpr LogStatementMetadata
        meta_data{F{}}(), Span{type_descriptors}};
    return &meta_data;
}
```

Metadata Aggregation: Metadata Generation

```
template <>
struct GetTypeDescriptor<int> {
    static constexpr TypeDescriptor value{Int{}}};
};

template <class F, class... Args>
LogStatementMetadata const* get_meta_data_ptr() {
    static constexpr std::array
        type_descriptors{GetTypeDescriptor<Args>::value...};
    static constexpr LogStatementMetadata
        meta_data{F{}}(), Span{type_descriptors}};
    return &meta_data;
}
```

Metadata Aggregation: Metadata Generation

```
template <>
struct GetTypeDescriptor<int> {
    static constexpr TypeDescriptor value{Int{}}};
};

template <class F, class... Args>
LogStatementMetadata const* get_meta_data_ptr() {
    static constexpr std::array
        type_descriptors{GetTypeDescriptor<Args>::value...};
    static constexpr LogStatementMetadata
        meta_data{F{()}, Span{type_descriptors}};
    return &meta_data;
}
```

Metadata Aggregation: Metadata Generation

```
template <>
struct GetTypeDescriptor<int> {
    static constexpr TypeDescriptor value{Int{}}};
};

template <class F, class... Args>
LogStatementMetadata const* get_meta_data_ptr() {
    static constexpr std::array
        type_descriptors{GetTypeDescriptor<Args>::value...};
    static constexpr LogStatementMetadata
        meta_data{F{ }(), Span{type_descriptors}};
    return &meta_data;
}
```

Metadata Aggregation: Metadata Generation

```
template <>
struct GetTypeDescriptor<int> {
    static constexpr TypeDescriptor value{Int{}}};
};

template <class F, class... Args>
LogStatementMetadata const* get_meta_data_ptr() {
    static constexpr std::array
        type_descriptors{GetTypeDescriptor<Args>::value...};
    static constexpr LogStatementMetadata
        meta_data{F{()}, Span{type_descriptors}};
    return &meta_data;
}
```

Metadata Aggregation: Metadata Generation C++ 20

```
struct Enum{  
    std::vector<std::pair<int, std::string_view>> table;};  
using TypeDescriptor = std::variant<Int, Float, Enum/*, ...*/>;  
using TypeDescriptors = std::vector<TypeDescriptor>;  
  
template <class F, class... Args>  
LogStatementMetadata const* get_meta_data_ptr() {  
    static constexpr LogStatementMetadata  
        meta_data{F{}}(),  
        std::vector{GetTypeDescriptor<Args>::value...}};  
    return &meta_data;  
}
```

- constexpr std::vector

Transporting Log Statements: Explanation<>

- Pointer to LogDataNode
 - ID
 - Metadata pointer
- Container of bytes
 - Serialized dynamic data

```
template <size_t size>
struct Explanation{
    LogDataNode* node{};
    std::array<std::byte, size> data{};
};
```


Transporting Log Statements: Usage

```
Expected<int32_t, Explanation<4U>> foo(int32_t y);
```

```
std::optional<int32_t> baz(int32_t y) {  
    if(auto x = foo(y); result.has_value()) {  
        return x.value(); }  
    else {  
        LOG_ERROR("baz failed due to error {error}",  
                  x.error()); }  
    return std::nullopt;  
}
```

Transporting Log Statements: Usage

```
Expected<int32_t, Explanation<4U>> foo(int32_t y);
```



```
std::optional<int32_t> baz(int32_t y) {  
    if(auto x = foo(y); result.has_value()) {  
        return x.value();  
    }  
    else {  
        LOG_ERROR("baz failed due to error {error}",  
                  x.error());  
    }  
    return std::nullopt;  
}
```

Transporting Log Statements: Usage

```
Expected<int32_t, Explanation<4U>> foo(int32_t y);
```

```
std::optional<int32_t> baz(int32_t y) {  
    if(auto x = foo(y); result.has_value()) {  
        return x.value(); }  
    else {  
        LOG_ERROR("baz failed due to error {error}",  
                  x.error()); }  
    return std::nullopt;  
}
```

Transporting Log Statements: Usage

```
Expected<int32_t, Explanation<4U>> foo(int32_t y);
```

```
std::optional<int32_t> baz(int32_t y) {  
    if(auto x = foo(y); result.has_value()) {  
        return x.value(); }  
    else {
```

```
        LOG_ERROR("baz failed due to error {error}",  
                  x.error()); }
```

```
    return std::nullopt;  
}
```

- **Client Decides when to log.**
- **Nesting creates traces.**

Conclusion

- Described a debug logger that:
 - Produces structured and text output
 - Separates meta and instance data for efficiency
- Implemented key pieces of the framework in C++17
 - Showed improvements in C++20
- Created a type that efficiently encodes and transports logging statements

Minimal Structured Logging for Autonomous Vehicles



Date: 09/18/2019

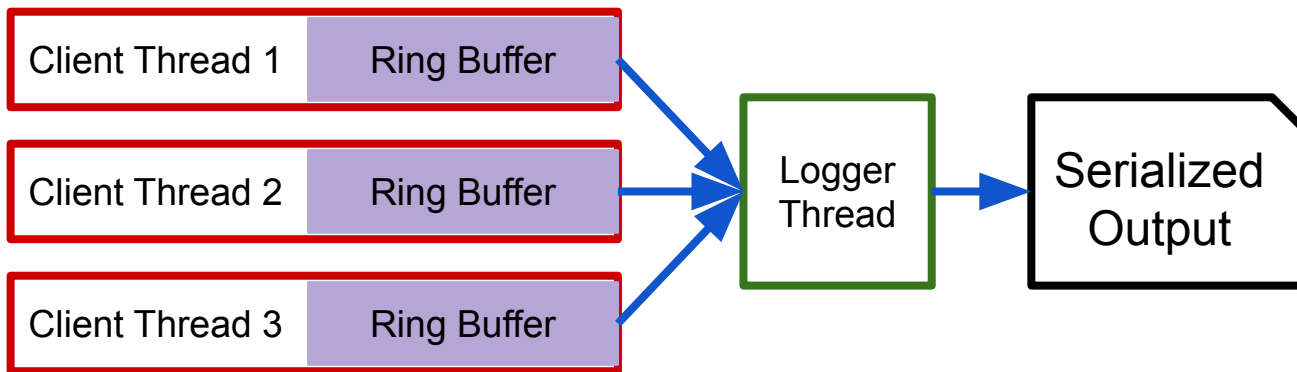
Prepared by: Robert Keelan
Email: rkeelan@argo.ai



Structured Payload: Dynamic Data

```
LOG_ERROR("Tried to access index {{index}} on vector of size  
{{size}}.", idx, siz);
```

- Description of each replacement field is contained in the metadata
- Replacement fields can just be serialized to an output archive
 - Archive could wrap a SPSC thread local queue



This Framework vs NanoLog

- NanoLog uses printf format string
- Less flexible TypeDescriptors
- Metadata generation
 - Separate preprocessor
 - **C++17 version writes metadata at first usage**