Unified Logging and Activity Tracing
Logging for the future
Session 721

Steven Szymanski Core OS Engineering
Matthieu Lucas System Applications
Agenda

Introduction
Logging Concepts
Demo
Using the Unified System
Tools
Best Practices
Gathering Logs
Deprications
Introduction
Background
Background

In 2014 Apple introduced Activity Tracing
In 2014 Apple introduced Activity Tracing.

We also introduced the concept of Faults and Errors.
Background

In 2014 Apple introduced Activity Tracing
We also introduced the concept of Faults and Errors
We recognize that Apple has several logging APIs
Goals
Goals

One common, efficient logging mechanism for both user and kernel mode
Goals

One common, efficient logging mechanism for both user and kernel mode
Maximize information collected while minimizing observer effect
Goals

One common, efficient logging mechanism for both user and kernel mode

Maximize information collected while minimizing observer effect

• Compressing data
Goals

One common, efficient logging mechanism for both user and kernel mode
Maximize information collected while minimizing observer effect
• Compressing data
• Deferring work and data collection
Goals

One common, efficient logging mechanism for both user and kernel mode
Maximize information collected while minimizing observer effect

• Compressing data
• Deferring work and data collection
• Managing log message lifecycle
Goals

One common, efficient logging mechanism for both user and kernel mode
Maximize information collected while minimizing observer effect
• Compressing data
• Deferring work and data collection
• Managing log message lifecycle
We want as much logging on all the time as possible
Goals

One common, efficient logging mechanism for both user and kernel mode
Maximize information collected while minimizing observer effect

• Compressing data
• Deferring work and data collection
• Managing log message lifecycle

We want as much logging on all the time as possible
Design privacy into the system
Features
Features

Improved categorization and filtering of log messages
Features

Improved categorization and filtering of log messages
Logging system collects caller information for you
Features

Improved categorization and filtering of log messages
Logging system collects caller information for you
New built-in type specifiers - simplifies log message preparation
Features

Improved categorization and filtering of log messages
Logging system collects caller information for you
New builtin type specifiers - simplifies log message preparation
New Console application and command-line tool
Features

Improved categorization and filtering of log messages
Logging system collects caller information for you
New built-in type specifiers - simplifies log message preparation
New Console application and command-line tool
Supported on macOS, iOS, tvOS, watchOS, and Simulators
Features

Improved categorization and filtering of log messages
Logging system collects caller information for you
New built-in type specifiers - simplifies log message preparation
New Console application and command-line tool
Supported on macOS, iOS, tvOS, watchOS, and Simulators
Support for Objective-C, C++ and C
Features

Improved categorization and filtering of log messages
Logging system collects caller information for you
New built-in type specifiers - simplifies log message preparation
New Console application and command-line tool
Supported on macOS, iOS, tvOS, watchOS, and Simulators
Support for Objective-C, C++ and C
Swift support in upcoming seed
Current Console
Console Revisited…
Console Revisited...
Logging Concepts
Adoption
Adoption

If you want to use the new Unified Logging system

• Build with the macOS 10.12, iOS 10.0, tvOS 10.0 or watchOS 3.0 SDK
• Legacy APIs (NSLog, asl_log_message, syslog…) redirected into new system
• Log data will be in new format and location
Adoption

If you want to use the new Unified Logging system

• Build with the macOS 10.12, iOS 10.0, tvOS 10.0 or watchOS 3.0 SDK
• Legacy APIs (NSLog, asl_log_message, syslog…) redirected into new system
• Log data will be in new format and location

If you don’t want to use the new Unified Logging system

• Build with macOS10.11, iOS 9.0, tvOS 9.0 and watchOS 2.0 SDK
• No changes
New File Formats
New File Formats

Log data is kept in a compressed binary format: .tracev3 files
New File Formats

Log data is kept in a compressed binary format: .tracev3 files
Stored under /var/db/diagnostics/ with support in /var/db/uuidtext
New File Formats

Log data is kept in a compressed binary format: .tracev3 files
Stored under /var/db/diagnostics/ with support in /var/db/uuidtext
New tools to access the stored and live log messages
New File Formats

Log data is kept in a compressed binary format: .tracev3 files
Stored under /var/db/diagnostics/ with support in /var/db/uuidtext
New tools to access the stored and live log messages
• Because data is stored in binary format, you MUST use new tools to access files
New File Formats

Log data is kept in a compressed binary format: .tracev3 files
Stored under /var/db/diagnostics/ with support in /var/db/uuidtext
New tools to access the stored and live log messages
• Because data is stored in binary format, you MUST use new tools to access files
New .logarchive format for portability of logs
New File Formats

Log data is kept in a compressed binary format: .tracev3 files
Stored under /var/db/diagnostics/ with support in /var/db/uuidtext
New tools to access the stored and live log messages
• Because data is stored in binary format, you MUST use new tools to access files
New .logarchive format for portability of logs
Subsystems and Categories
Subsystems and Categories

Log messages can be associated with a subsystem and category
Subsystems and Categories

Log messages can be associated with a subsystem and category.

Can be used to control how log messages are filtered and displayed.
Subsystems and Categories

Log messages can be associated with a subsystem and category
Can be used to control how log messages are filtered and displayed
A subsystem can contain multiple categories
Subsystems and Categories

Log messages can be associated with a subsystem and category.
Can be used to control how log messages are filtered and displayed.
A subsystem can contain multiple categories.
You can use as many subsystems and categories as needed.
Log messages can be associated with a subsystem and category. Can be used to control how log messages are filtered and displayed. A subsystem can contain multiple categories. You can use as many subsystems and categories as needed.

Example:
Subsystems and Categories

Log messages can be associated with a subsystem and category.
Can be used to control how log messages are filtered and displayed.
A subsystem can contain multiple categories.
You can use as many subsystems and categories as needed.
Example:

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.your-company.your-application</td>
<td>setup, inprogress, teardown</td>
</tr>
</tbody>
</table>
Subsystems and Categories

Log messages can be associated with a subsystem and category
Can be used to control how log messages are filtered and displayed
A subsystem can contain multiple categories
You can use as many subsystems and categories as needed
Example:

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.your-company.your-application</td>
<td>setup, inprogress, teardown</td>
</tr>
<tr>
<td>com.your-company.test.your-application</td>
<td>test</td>
</tr>
</tbody>
</table>
Logging Behavior
Logging Behavior

Each log message has a level determined by the API used.
Logging Behavior

Each log message has a level determined by the API used

- Three basic levels—Default, Info, Debug
Logging Behavior

Each log message has a level determined by the API used

- Three basic levels—Default, Info, Debug
- Two special levels—Fault, Error
Logging Behavior

Each log message has a level determined by the API used

- Three basic levels—Default, Info, Debug
- Two special levels—Fault, Error

Each basic level has two characteristics that can be set for system, subsystem, or category
Logging Behavior

Each log message has a level determined by the API used

• Three basic levels—Default, Info, Debug
• Two special levels—Fault, Error

Each basic level has two characteristics that can be set for system, subsystem, or category

• Is is enabled? (Default messages are always enabled)
Logging Behavior

Each log message has a level determined by the API used

• Three basic levels—Default, Info, Debug
• Two special levels—Fault, Error

Each basic level has two characteristics that can be set for system, subsystem, or category

• Is is enabled? (Default messages are always enabled)
• Is it stored to disk or memory?
Logging Behavior

Each log message has a level determined by the API used

- Three basic levels—Default, Info, Debug
- Two special levels—Fault, Error

Each basic level has two characteristics that can be set for system, subsystem, or category

- Is it enabled? (Default messages are always enabled)
- Is it stored to disk or memory?

The levels are hierarchical
Logging Behavior

Each log message has a level determined by the API used

• Three basic levels—Default, Info, Debug
• Two special levels—Fault, Error

Each basic level has two characteristics that can be set for system, subsystem, or category

• Is it enabled? (Default messages are always enabled)
• Is it stored to disk or memory?

The levels are hierarchical

• So setting Debug to go to disk implies that Info will also go to disk
Logging Behavior

Each log message has a level determined by the API used

- Three basic levels—Default, Info, Debug
- Two special levels—Fault, Error

Each basic level has two characteristics that can be set for system, subsystem, or category

- Is is enabled? (Default messages are always enabled)
- Is it stored to disk or memory?

The levels are hierarchical

- So setting Debug to go to disk implies that Info will also go to disk

Behavior can be customized by installing profiles or, on macOS, via log command
Standard Behavior
# Standard Behavior

<table>
<thead>
<tr>
<th>Message Level</th>
<th>Enabled</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT LEVEL</td>
<td>ALWAYS</td>
<td>DISK</td>
</tr>
<tr>
<td>INFO LEVEL</td>
<td>YES</td>
<td>MEMORY</td>
</tr>
<tr>
<td>DEBUG LEVEL</td>
<td>NO</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Standard Behavior

<table>
<thead>
<tr>
<th>Message Level</th>
<th>Enabled</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT LEVEL</td>
<td>ALWAYS</td>
<td>DISK</td>
</tr>
<tr>
<td>INFO LEVEL</td>
<td>YES</td>
<td>MEMORY</td>
</tr>
<tr>
<td>DEBUG LEVEL</td>
<td>NO</td>
<td>N/A</td>
</tr>
<tr>
<td>ERROR</td>
<td>ALWAYS</td>
<td>DISK</td>
</tr>
<tr>
<td>FAULT</td>
<td>ALWAYS</td>
<td>DISK</td>
</tr>
</tbody>
</table>
Privacy

Prevent accidental logging of Personally Identifiable Information (PII)
Privacy

Prevent accidental logging of Personally Identifiable Information (PII)
Dynamic strings, collections, arrays, etc. are assumed to be private
Faults and Errors
Faults and Errors

We do extra work saving additional information on Fault or Error
Faults and Errors

We do extra work saving additional information on Fault or Error

Errors represent issues discovered within a given application/library
Faults and Errors

We do extra work saving additional information on Fault or Error.
Errors represent issues discovered within a given application/library.
Faults represent more global problems in the system.
Faults and Errors

We do extra work saving additional information on Fault or Error

Errors represent issues discovered within a given application/library

Faults represent more global problems in the system

Faults and Error log information is captured into a separate set of log files
Architecture
Architecture

Process A

Buffer

Process B

Buffer

4k

Buffer
Architecture

- **Process A**
  - Buffer

- **Process B**
  - Buffer

- **Compressor**
  - Compressed Buffer

- **logd**
  - Memory Only Buffer

- **diagnosticd**
  - "Live" log stream
  - Compressed Log Files
  - Regular Log Data
  - Fault and Error Log Data
  - Others
Profile can change routing and rules for given applications or subsystems.
Demo

Console demo
Using the Unified System
## Summary of New APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Destination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>os_log</td>
<td>Disk</td>
<td>Default logging level that is always captured</td>
</tr>
<tr>
<td>os_log_info</td>
<td>Memory</td>
<td>Additional information (defaults to memory-only buffers)</td>
</tr>
<tr>
<td>os_log_debug</td>
<td>Off</td>
<td>Debug level content (off-by default)</td>
</tr>
<tr>
<td>os_log_error</td>
<td>Disk</td>
<td>Process local error</td>
</tr>
<tr>
<td>os_log_fault</td>
<td>Disk</td>
<td>System-level error (usually involves multiple processes)</td>
</tr>
<tr>
<td>os_log_create</td>
<td>n/a</td>
<td>Create a log object for custom behaviors</td>
</tr>
</tbody>
</table>
Creating a Log Object
Creating a Log Object

```c
os_log_t log = os_log_create("com.your_company.subsystem", "network");
```

Create thread-safe singleton object that controls behavior of log messages
Creating a Log Object

```c
os_log_t log = os_log_create("com.your_company.subsystem", "network");
```

Create thread-safe singleton object that controls behavior of log messages

Defaults to system-behavior
Creating a Log Object

```c
os_log_t log = os_log_create("com.your_company.subsystem", "network");
```

Create thread-safe singleton object that controls behavior of log messages

Defaults to system-behavior

Usage

```c
os_log(log, "This happened");
```
Creating a Log Object

```c
os_log_t log = os_log_create("com.your_company.subsystem", "network");
```

Create thread-safe singleton object that controls behavior of log messages

Defaults to system-behavior

Usage

```c
os_log(log, "This happened");
```

Reference to category and subsystem stored with every log message
Creating a Log Object

```c
os_log_t log = os_log_create("com.your_company.subsystem", "network");
```

Create thread-safe singleton object that controls behavior of log messages

Defaults to system-behavior

Usage

```c
os_log(log, "This happened");
```

Reference to category and subsystem stored with every log message

Or use `OS_LOG_DEFAULT`

- For messages not associated with subsystem/category
Creating a Log Object

```c
os_log_t log = os_log_create("com.your_company.subsystem", "network");
```
Create thread-safe singleton object that controls behavior of log messages

Defaults to system-behavior

Usage

```c
os_log(log, "This happened");
```
Reference to category and subsystem stored with every log message

Or use `OS_LOG_DEFAULT`

- For messages not associated with subsystem/category
Built-in Type Formatters

We all spend too much code converting binary information to strings to log
Built-in Type Formatters

We all spend too much code converting binary information to strings to log

Built-in decoding for common values

"%{time_t}d" or "%{errno}d"
Built-in Type Formatters

We all spend too much code converting binary information to strings to log.

Built-in decoding for common values

```
"%{time_t}d" or "%{errno}d"
```

Arbitrary binary data using a new format type

```
"%.P"
```
Built-in Type Formatters

We all spend too much code converting binary information to strings to log

Built-in decoding for common values

```
"%{time_t}d" or "%{errno}d"
```

Arbitrary binary data using a new format type

```
"%.16P"
```

Built-in decoding for binary-types

```
"%{uuid_t}.16P"
```
<table>
<thead>
<tr>
<th>Type</th>
<th>Format String</th>
<th>Example Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_t</td>
<td>%{time_t}d</td>
<td>2016-01-12 19:41:37</td>
</tr>
<tr>
<td>timeval</td>
<td>%{timeval}.*P</td>
<td>2016-01-12 19:41:37.774236</td>
</tr>
<tr>
<td>timespec</td>
<td>%{timespec}.*P</td>
<td>2016-01-12 19:41:37.774236823</td>
</tr>
<tr>
<td>errno</td>
<td>%{errno}d</td>
<td>Broken pipe</td>
</tr>
<tr>
<td>uuid_t</td>
<td>%{uuid_t}.16P</td>
<td>10742E39-0657-41F8-AB99-878C5EC2DCAA</td>
</tr>
<tr>
<td></td>
<td>%{uuid_t}.*P</td>
<td></td>
</tr>
<tr>
<td>sockaddr</td>
<td>%{network:sockaddr}.*P</td>
<td>fe80::f:86ff:fee9:5c16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17.43.23.87</td>
</tr>
<tr>
<td>in_addr</td>
<td>%{network:in_addr}d</td>
<td>17.43.23.87</td>
</tr>
<tr>
<td>in6_addr</td>
<td>%{network:in6_addr}.16P</td>
<td>fe80::f:86ff:fee9:5c16</td>
</tr>
</tbody>
</table>
Per Parameter Privacy
Per Parameter Privacy

Privacy is handled on a parameter by parameter basis
Per Parameter Privacy

Privacy is handled on a parameter by parameter basis.
Scalars and static strings are assumed to be public.
Per Parameter Privacy

Privacy is handled on a parameter by parameter basis
Scalar and static strings are assumed to be public
Dynamic strings, collections, and objects are assumed to be private
Per Parameter Privacy

Privacy is handled on a parameter by parameter basis
 Scalars and static strings are assumed to be public
 Dynamic strings, collections, and objects are assumed to be private
 Can be overridden on a per-parameter basis

"%{public}@" or "%{private}d"
Per Parameter Privacy

Privacy is handled on a parameter by parameter basis.

Scalars and static strings are assumed to be public.

Dynamic strings, collections, and objects are assumed to be private.

Can be overridden on a per-parameter basis.

"{%public}@" or "{%private}d"

Combine privacy and formatting:

"{%public, uuid_t}.16P"
/*
 * Log Message Simplification
 */
/*
 * Log Message Simplification
 */

// Old way:
/*
 * Log Message Simplification
 */

// Old way:
if (LogLevelEnabled(info)) {

/**
 * Log Message Simplification
 */

// Old way:
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
/*
 * Log Message Simplification
 */

// Old way:
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = __convert_sockaddr(&sa);
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = _convert_sockaddr(&sa);
    NSLog(@"%s (%s:%d) - fd: %d, uuid: %s, IP: %s",
        __PRETTY_FUNCTION__, __FILE__, __LINE__, fd, uuid_str, addr_desc);
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = _convert_sockaddr(&sa);
    NSLog(@"%s (%s:%d) - fd: %d, uuid: %s, IP: %s",
         __PRETTY_FUNCTION__, __FILE__, __LINE__, fd, uuid_str, addr_desc);
    free(addr_desc);
}
/*
 * Log Message Simplification
 */

// Old way:
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = _convert_sockaddr(&sa);
    NSLog(@"%s (%s:%d) - fd: %d, uuid: %s, IP: %s",
         __PRETTY_FUNCTION__, __FILE__, __LINE__, fd, uuid_st, addr_desc);
    free(addr_desc);
}

/*
 * Log Message Simplification
 */

// Old way:
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = _convert_sockaddr(&sa);
    NSLog(@"%s (%s:%d) - fd: %d, uuid: %s, IP: %s",
        __PRETTY_FUNCTION__, __FILE__, __LINE__, fd, uuid_str, addr_desc);
    free(addr_desc);
}

// New way:
/*
 * Log Message Simplification
 */

// Old way:
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = _convert_sockaddr(&sa);
    NSLog(@"%s (%s:%d) - fd: %d, uuid: %s, IP: %s",
         __PRETTY_FUNCTION__, __FILE__, __LINE__, fd, uuid_str, addr_desc);
    free(addr_desc);
}

// New way:
os_log_info(OS_LOG_DEFAULT,
// Old way:
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = _convert_sockaddr(&sa);
    NSLog(@"%s (%s:%d) - fd: %d, uuid: %s, IP: %s",
        __PRETTY_FUNCTION__, __FILE__, __LINE__, fd, uuid_st, addr_desc);
    free(addr_desc);
}

// New way:
os_log_info(OS_LOG_DEFAULT,
    "fd: %d, uuid: %{uuid_t}.16P, IP: %{network:sockaddr}.P",
/*
 * Log Message Simplification
 */

// Old way:
if (LogLevelEnabled(info)) {
    uuid_string_t uuid_str;
    uuid_unparse_upper(uuid, uuid_str);
    char *addr_desc = _convert_sockaddr(&sa);
    NSLog(@"%s (%s:%d) - fd: %d, uuid: %s, IP: %s",
         __PRETTY_FUNCTION__, __FILE__, __LINE__, fd, uuid_str, addr_desc);
    free(addr_desc);
}

// New way:
os_log_info(OS_LOG_DEFAULT,
    "fd: %d, uuid: %{uuid_t}.16P, IP: %{network:sockaddr}.*P",
    fd, uuid, sa->sa_len, &sa);
/*
 * Example Code
 */
os_log_t general_log = os_log_create("com.apple.logging.example", "general");
/*
 * Example Code
 */

os_log_t general_log = os_log_create("com.apple.logging.example", "general");

os_log_t time_log = os_log_create("com.apple.logging.example", "timestamp");
os_log_t general_log = os_log_create("com.apple.logging.example", "general");

os_log_t time_log = os_log_create("com.apple.logging.example", "timestamp");

os_log(general_log, "running example code");
/*
 * Example Code
 */

os_log_t general_log = os_log_create("com.apple.logging.example", "general");

os_log_t time_log = os_log_create("com.apple.logging.example", "timestamp");

os_log(general_log, "running example code");

os_log_info(general_log, "processing file %\{public\}s", filename);
/*
 * Example Code
 */

os_log_t general_log = os_log_create("com.apple.logging.example", "general");

os_log_t time_log = os_log_create("com.apple.logging.example", "timestamp");

os_log(general_log, "running example code");

os_log_info(general_log, "processing file %{public}s", filename);

int fd = open(filename, O_RDONLY);
if (fd < 0) {
    os_log_error(general_log, "Cannot open file %{public}s - %{errno}d", filename, errno);
    ...
}
os_log_t general_log = os_log_create("com.apple.logging.example", "general");
os_log_t time_log = os_log_create("com.apple.logging.example", "timestamp");

os_log(general_log, "running example code");

os_log_info(general_log, "processing file %{public}s", filename);

int fd = open(filename, O_RDONLY);
if (fd < 0) {
    os_log_error(general_log, "Cannot open file %{public}s - %{errno)d", filename, errno);
    . . .
}

struct stat sb;
if (fstat(fd, &sb) < 0) {
    os_log_fault(general_log, "Failed to fstat %{public}s - %{errno)d", filename, errno);
    . . .
}
/*
 * Example Code
 */

os_log_t general_log = os_log_create("com.apple.logging.example", "general");

os_log_t time_log = os_log_create("com.apple.logging.example", "timestamp");

os_log(general_log, "running example code");

os_log_info(general_log, "processing file %{public}s", filename);

int fd = open(filename, O_RDONLY);
if (fd < 0) {
    os_log_error(general_log, "Cannot open file %{public}s - %{errno}d", filename, errno);
    ..
}

struct stat sb;
if (fstat(fd, &sb) < 0) {
    os_log_fault(general_log, "Failed to fstat %{public}s - %{errno}d", filename, errno);
    ..
}

os_log_info(time_log,
    "status for file %{public}s, atime:%{timespec}.*P, mtime:%{timespec}.*P, ctime:%{timespec}.*P",
    filename,
    sizeof(struct timespec), &sb.st_atimespec,
    sizeof(struct timespec), &sb.st_mtimespec,
    sizeof(struct timespec), &sb.st_ctimespec);
Activity API Improvements
Activity API Improvements

Activities are now objects that can be stored and re-used
Activity API Improvements

Activities are now objects that can be stored and re-used

- Direct control of activity relationships during creation
Activity API Improvements

Activities are now objects that can be stored and re-used

- Direct control of activity relationships during creation

New API to auto-scope activities within your code
# Improved Activity APIs

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>os_activity_create</td>
<td>Creates a new activity object</td>
</tr>
<tr>
<td>os_activity_scope</td>
<td>Makes an execution block a part of an activity</td>
</tr>
<tr>
<td>os_activity_apply</td>
<td>Invokes a block scoped to a given activity</td>
</tr>
<tr>
<td>os_activity_label_useraction</td>
<td>Label an activity as a user action (UI-based activities)</td>
</tr>
</tbody>
</table>
/*
 * New Activity API Example
 */
os_activity_t init_activity = os_activity_create("Init", OS_ACTIVITY_CURRENT, OS_ACTIVITY_FLAG_DEFAULT);
New Activity API Example

```c
os_activity_t init_activity = os_activity_create("Init", OS_ACTIVITY_CURRENT,
OS_ACTIVITY_FLAG_DEFAULT);

os_activity_t verify_activity = os_activity_create("Verify", init_activity,
OS_ACTIVITY_FLAG_DEFAULT);
```
/*
 * New Activity API Example
 */

os_activity_t init_activity = os_activity_create("Init", OS_ACTIVITY_CURRENT,
OS_ACTIVITY_FLAG_DEFAULT);

os_activity_t verify_activity = os_activity_create("Verify", init_activity,
OS_ACTIVITY_FLAG_DEFAULT);

if (isReady) {
    os_activity_scope(verify_activity);
    // All of the following work is done under “verification activity scope”
    . . .
}

New Activity API Example

```c
os_activity_t init_activity = os_activity_create("Init", OS_ACTIVITY_CURRENT, OS_ACTIVITY_FLAG_DEFAULT);

os_activity_t verify_activity = os_activity_create("Verify", init_activity, OS_ACTIVITY_FLAG_DEFAULT);

if (isReady) {
    os_activity_scope(verify_activity);
    // All of the following work is done under “verification activity scope”
    ...
}

// This is not part of that activity
```
/*
 * New Activity API Example
 */

os_activity_t init_activity = os_activity_create("Init", OS_ACTIVITY_CURRENT, OS_ACTIVITY_FLAG_DEFAULT);

os_activity_t verify_activity = os_activity_create("Verify", init_activity, OS_ACTIVITY_FLAG_DEFAULT);

if (isReady) {
    os_activity_scope(verify_activity);
    // All of the following work is done under "verification activity scope"
    ...
}
// This is not part of that activity

os_activity_apply(init_activity, ^{
    // do some work based on the "init_activity"
    ...}
);
Tools
Console
Console

View live content from a system
Console

View live content from a system
Open log archives
Console

View live content from a system
Open log archives
New Activity centric view of logging and tracing
Console

View live content from a system
Open log archives
New Activity centric view of logging and tracing
Advanced filtering and searching
Console

View live content from a system
Open log archives
New Activity centric view of logging and tracing
Advanced filtering and searching
Device support
log Command Line Tool
log Command Line Tool

Same functionality as Console from the command line
log Command Line Tool

Same functionality as Console from the command line
Stream live log messages
log Command Line Tool

Same functionality as Console from the command line
Stream live log messages

$ log stream
log Command Line Tool

Same functionality as Console from the command line

Stream live log messages

$ log stream
$ log stream --predicate 'eventMessage contains "my message"'
log Command Line Tool

Same functionality as Console from the command line

Stream live log messages

$ log stream
$ log stream --predicate 'eventMessage contains "my message"'

Display a log file or archive
log Command Line Tool

Same functionality as Console from the command line

Stream live log messages

$ log stream
$ log stream --predicate 'eventMessage contains "my message"'

Display a log file or archive

$ log show system_logs.logarchive
log Command Line Tool

Same functionality as Console from the command line

Stream live log messages

```
$ log stream
$ log stream --predicate 'eventMessage contains "my message"'
```

Display a log file or archive

```
$ log show system_logs.logarchive
```

Enable debug for your subsystem on macOS
log Command Line Tool

Same functionality as Console from the command line

Stream live log messages

$ log stream
$ log stream --predicate 'eventMessage contains "my message"'

Display a log file or archive

$ log show system_logs.logarchive

Enable debug for your subsystem on macOS

$ log config --mode "level:debug" --subsystem com.mycorp.myapp
log Command Line Tool

Same functionality as Console from the command line
Stream live log messages

```bash
$ log stream
$ log stream --predicate 'eventMessage contains "my message"'
```
Display a log file or archive

```bash
$ log show system_logs.logarchive
```
Enable debug for your subsystem on macOS

```bash
$ log config --mode "level:debug" --subsystem com.mycorp.myapp
```
Coming Soon
Coming Soon

Tools for accessing new log information from 10.11 are coming soon
Coming Soon

Tools for accessing new log information from 10.11 are coming soon

But... in the meantime
Coming Soon

Tools for accessing new log information from 10.11 are coming soon

But... in the meantime

$ xcrun simctl spawn booted log show system_logs.logarchive
Coming Soon

Tools for accessing new log information from 10.11 are coming soon

But... in the meantime

$ xcrun simctl spawn booted log show system_logs.logarchive
Best Practices
Logging Etiquette
Logging Etiquette

Ensure messages contain only information useful for debugging
Logging Etiquette

Ensure messages contain only information useful for debugging

Let us do the formatting for you—leverage built-in formatters
Logging Etiquette

Ensure messages contain only information useful for debugging
Let us do the formatting for you—leverage built-in formatters
Avoid creating wrapper functions for os_log* APIs
Logging Etiquette

Ensure messages contain only information useful for debugging
Let us do the formatting for you—leverage built-in formatters
Avoid creating wrapper functions for os_log* APIs
Log only what you need from collections (Dictionaries, Arrays, etc.)
Logging Etiquette

Ensure messages contain only information useful for debugging
Let us do the formatting for you—leverage built-in formatters
Avoid creating wrapper functions for `os_log*` APIs
Log only what you need from collections (Dictionaries, Arrays, etc.)
Avoid logging in tight code loops
Using os_log Family of APIs
Using os_log Family of APIs

Use `os_log` to log critical details to help debug issues
Using os_log Family of APIs

Use `os_log` to log critical details to help debug issues

Use `os_log_info` for additional info that will be captured during error or fault
Using **os_log** Family of APIs

Use **os_log** to log critical details to help debug issues

Use **os_log_info** for additional info that will be captured during error or fault

Use **os_log_debug** for high-volume debugging during development
Using os_log Family of APIs

Use **os_log** to log critical details to help debug issues

Use **os_log_info** for additional info that will be captured during error or fault

Use **os_log_debug** for high-volume debugging during development

Use **os_log_error** to cause additional information capture from app
Using os_log Family of APIs

Use `os_log` to log critical details to help debug issues

Use `os_log_info` for additional info that will be captured during error or fault

Use `os_log_debug` for high-volume debugging during development

Use `os_log_error` to cause additional information capture from app

Use `os_log_fault` to cause additional information capture from system
Gathering Logs
Using sysdiagnose
Using sysdiagnose

sysdiagnose is the preferred method to capture data for bug reports
Using sysdiagnose

sysdiagnose is the preferred method to capture data for bug reports

• Unified Logging data in system_logs.archive
Using sysdiagnose

sysdiagnose is the preferred method to capture data for bug reports

- Unified Logging data in system_logs.archive

You can use key-chord to trigger
Using sysdiagnose

sysdiagnose is the preferred method to capture data for bug reports

- Unified Logging data in system_logs.archive

You can use key-chord to trigger

sysdiagnose on Apple Watch will trigger on both Apple Watch and iPhone
Using sysdiagnose

sysdiagnose is the preferred method to capture data for bug reports

- Unified Logging data in system_logs.archive

You can use key-chord to trigger

sysdiagnose on Apple Watch will trigger on both Apple Watch and iPhone

Transfer from device using iTunes
Using sysdiagnose

sysdiagnose is the preferred method to capture data for bug reports
• Unified Logging data in system_logs.archive
You can use key-chord to trigger
sysdiagnose on Apple Watch will trigger on both Apple Watch and iPhone
Transfer from device using iTunes
This is the file to send to Apple (either Radar or Developer Technical Support)
# Key-chords for sysdiagnose

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS</td>
<td>Shift + Control + Option + Command + Period (.)</td>
</tr>
</tbody>
</table>
| iOS     | Volume Up + Volume Down + Power  
|         | Slight vibration on iPhone to indicate start |
| watchOS | Press and hold Digital Crown + Side Button for 1 second  
|         | A screen shot is triggered if not held long enough. Slight haptic to indicate start. |
| tvOS    | Play/Pause + Volume Down  
|         | On older remote controls for Apple TV, must be held for 5 seconds |
Deprecations
Deprecation of Legacy logging APIs
Deprecation of Legacy logging APIs

All ASL logging APIs are superseded by the new APIs.
Deprecation of Legacy logging APIs

All ASL logging APIs are superseded by the new APIs

New APIs for searching new log data will not be made public this release
Deprecation of Legacy logging APIs

All ASL logging APIs are superseded by the new APIs

New APIs for searching new log data will not be made public this release

- No equivalent asl_search functionality
## Deprecated Activity APIs

<table>
<thead>
<tr>
<th>Function</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>os_activity_start</code></td>
<td>Use <code>os_activity_create</code> and <code>os_activity_scope</code> / <code>os_activity_apply</code></td>
</tr>
<tr>
<td><code>os_activity_end</code></td>
<td>Use <code>os_activity_create</code> and <code>os_activity_scope</code> / <code>os_activity_apply</code></td>
</tr>
<tr>
<td><code>os_activity_set_breadcrumb</code></td>
<td>Use <code>os_activity_label_useraction</code></td>
</tr>
<tr>
<td><code>os_trace_with_payload</code></td>
<td>Use <code>os_log*</code></td>
</tr>
</tbody>
</table>
Summary

The new Unified Logging system is
Summary

The new Unified Logging system is

• Faster
Summary

The new Unified Logging system is

- Faster
- Easier to use
Summary

The new Unified Logging system is

• Faster
• Easier to use
• Gives you more control
The new Unified Logging system is

- Faster
- Easier to use
- Gives you more control

But requires using new APIs and new tools
## Related Sessions

<table>
<thead>
<tr>
<th>Session Title</th>
<th>Event Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fix Bugs Faster Using Activity Tracing</td>
<td>WWDC 2014</td>
</tr>
</tbody>
</table>
More Information
