Parseable Driver Output

Introduction

This document serves to describe the parseable output format provided by the Swift compiler driver with the "-parseable-output" flag. This output format is intended to be parsed by other programs; one such use case is to allow an IDE to construct a detailed log based on the commands the driver issued.

Message Format

The parseable output provided by the Swift driver is provided as messages encoded in JSON objects. All messages are structured like this::

```
<Message Length>\n
{
  "kind": "<Message Kind>",
  "name": "<Message Name>",
  "<key>": "<value>",
  ...
}\n
```

This allows the driver to pass as much information as it wants about the ongoing compilation, and programs which parse this data can decide what to use and what to ignore.

Quasi-PIDs

In previous versions of this format, certain fields labeled "pid" carry only legitimate process IDs (PIDs) from the underlying operating system. These PIDs are used by the driver to associate events that happen to jobs, such as signals when a job crashes with the file being compiled during that event. Real PIDs are always positive numbers, in keeping with POSIX semantics and the unsigned Process ID type on Windows.

In more recent versions of this format, the Swift driver combines multiple jobs together into batches, running each batch in a single subprocess. In this mode, the driver writes messages describing each job as before, but assigns to each a so-called "quasi-PID" for the "pid" field, which does not correspond to an operating system process ID. Rather, a quasi-PID is a **negative number** that can be used **like** a PID, both as a unique identifier for each job that was run and in order to relate messages and tasks together, but it does not denote a real process. By using negative numbers for quasi-PIDs, we ensure that no two jobs will have the same PID: no batch mode job can have the same PID as a real process. Two jobs with the same PID would violate this uniqueness assumption.

However, if you need to know the real PID of the process, you can use the 'process.real_pid' field in the began/finished/signalled message. It will contain the process identifier the operating system gave the process. Be aware that this does not guarantee to be unique as contrasted with "pid"! If "pid" is greater than 0 it will always match the value in "process.real_pid".

Message Kinds

The driver may emit four kinds of messages: "began", "finished", "signalled", and "skipped".

Began Message

A "began" message indicates that a new task began. As with all task-based messages, it will include the task's PID (or quasi-PID) under the "pid" key. It may specify the task's inputs as an array of paths under the "inputs" key. It may specify the task's outputs as an array of objects under the "outputs" key. An "outputs" object will have two fields, a "kind" describing the type of the output, and a "path" containing the path to the output. A "began" message will specify the command which was executed under the "command_executable" and "command_arguments" keys. To get the real process identifier for the process that started, get the value of "process.real_pid".

```
{
  "kind": "began",
  "name": "compile",
  "pid": 12345,
  "process": {
    "real_pid": 12345
  },
  "inputs": ["/src/foo.swift"],
  "outputs": [
     {
       "type": "object",
       "path": "/build/foo.o"
     },
     {
       "type": "swiftmodule",
       "path": "/build/foo.swiftmodule"
     },
     {
       "type": "diagnostics",
       "path": "/build/foo.dia"
     },
  ],
  "command_executable": "swift",
```

```
  "command_arguments" : ["-frontend", "-c", "-primary-file", "/src/foo.swift", "/src/bar.swi
}
```

Finished Message

A "finished" message indicates that a task finished execution. As with all task-based messages, it will include the task's PID (or quasi-PID) under the "pid" key. It will include the exit status of the task under the "exit-status" key. It may include the stdout/stderr of the task under the "output" key; if this key is missing, no output was generated by the task. It will contain the process identifier of the operating system and usage under the "process" key. The usage is optional and could be omitted.

```
{
 "kind": "finished",
 "name": "compile",
 "pid": 12345,
 "exit-status": 0,
 "process": {
   "real_pid": 12345,
   "usage": {
     "utime": 22740,
     "stime": 91107,
     "maxrss": 7745536
   }
  }
 // "output" key omitted because there was no stdout/stderr.
}
```

Signalled Message

A "signalled" message indicates that a task exited abnormally due to a signal. As with all task-based message, it will include the task's PID (or quasi-PID) under the "pid" key. It may include an error message describing the signal under the "error-message" key. As with the "finished" message, it may include the stdout/stderr of the task under the "output" key; if this key is missing, no output was generated by the task. It may include the "signal" key, the terminating signal number. (This may not be available on all platforms.) It will contain the process identifier of the operating system and usage under the "process" key. The usage is optional and could be omitted.

Example::

```
{
 "kind": "signalled",
 "name": "compile",
 "pid": 12345,
```

```
 "error-message": "Segmentation fault: 11",
 "signal": 4,
 "process": {
   "real_pid": 12345,
   "usage": {
     "utime": 22740,
     "stime": 91107,
     "maxrss": 7745536
   }
  }
 // "output" key omitted because there was no stdout/stderr.
}
```

Skipped Message

A "skipped" message indicates that the driver determined a command did not need to run during the current compilation. A "skipped" message is equivalent to a "began" message, with the exception that it does not include the "pid" key.

```
{
 "kind": "skipped",
 "name": "compile",
 "inputs": ["/src/foo.swift"],
 "outputs": [
    {
      "type": "object",
      "path": "/build/foo.o"
    },
    {
      "type": "swiftmodule",
      "path": "/build/foo.swiftmodule"
    },
    {
      "type": "diagnostics",
      "path": "/build/foo.dia"
    },
 ],
 "command_executable": "swift",
 "command_arguments": ["-frontend", "-c", "-primary-file", "/src/foo.swift", "/src/bar.swift
}
```

Message Names

The name of the message identifies the kind of command the message describes. Some valid values are:

- compile
- merge-module
- link
- generate-dsym

A “compile” message represents a regular Swift frontend command. A “merge-module” message represents an invocation of the Swift frontend which is used to merge partial swiftmodule files into a complete swiftmodule. A “link” message indicates that the driver is invoking the linker to produce an executable or a library. A “generate-dsym” message indicates that the driver is invoking dsymutil to generate a dSYM.

Parsers of this format should be resilient in the event of an unknown name, as the driver may emit messages with new names whenever it needs to execute a new kind of command.