libuv 0.10 -> 1.0.0 migration guide

Some APIs changed quite a bit throughout the 1.0.0 development process. Here is a migration guide for the most significant changes that happened after 0.10 was released.

Loop initialization and closing

In libuv 0.10 (and previous versions), loops were created with *uv_loop_new*, which allocated memory for a new loop and initialized it; and destroyed with *uv_loop_delete*, which destroyed the loop and freed the memory. Starting with 1.0, those are deprecated and the user is responsible for allocating the memory and then initializing the loop.

libuv 0.10

```
uv_loop_t* loop = uv_loop_new();
...
uv_loop_delete(loop);
```

libuv 1.0

```
uv_loop_t* loop = malloc(sizeof *loop);
uv_loop_init(loop);
...
uv_loop_close(loop);
free(loop);
```

Note

Error handling was omitted for brevity. Check the documentation for :c:func:`uv_loop_init` and :c:func:`uv_loop_close`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, line 39); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, line 39); *backlink*

Unknown interpreted text role "c:func".

Error handling

Error handling had a major overhaul in libuv 1.0. In general, functions and status parameters would get 0 for success and -1 for failure on libuv 0.10, and the user had to use *uv_last_error* to fetch the error code, which was a positive number.

In 1.0, functions and status parameters contain the actual error code, which is 0 for success, or a negative number in case of error.

libuv 0.10

```
... assume 'server' is a TCP server which is already listening
r = uv_listen((uv_stream_t*) server, 511, NULL);
if (r == -1) {
  uv_err_t err = uv_last_error(uv_default_loop());
  /* err.code contains UV_EADDRINUSE */
}
```

libuv 1.0

```
... assume 'server' is a TCP server which is already listening
r = uv_listen((uv_stream_t*) server, 511, NULL);
if (r < 0) {
  /* r contains UV_EADDRINUSE */
}
```

Threadpool changes

In libuv 0.10 Unix used a threadpool which defaulted to 4 threads, while Windows used the *QueueUserWorkItem* API, which uses a Windows internal threadpool, which defaults to 512 threads per process.

In 1.0, we unified both implementations, so Windows now uses the same implementation Unix does. The threadpool size can be set by exporting the `UV_THREADPOOL_SIZE` environment variable. See :c:ref:`threadpool`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, **line 82**); ***backlink***
>
> Unknown interpreted text role "c:ref".

Allocation callback API change

In libuv 0.10 the callback had to return a filled :c:type:`uv_buf_t` by value:

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, **line 90**); ***backlink***
>
> Unknown interpreted text role "c:type".

```
uv_buf_t alloc_cb(uv_handle_t* handle, size_t size) {
    return uv_buf_init(malloc(size), size);
}
```

In libuv 1.0 a pointer to a buffer is passed to the callback, which the user needs to fill:

```
void alloc_cb(uv_handle_t* handle, size_t size, uv_buf_t* buf) {
    buf->base = malloc(size);
    buf->len = size;
}
```

Unification of IPv4 / IPv6 APIs

libuv 1.0 unified the IPv4 and IPv6 APIS. There is no longer a *uv_tcp_bind* and *uv_tcp_bind6* duality, there is only :c:func:`uv_tcp_bind` now.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, **line 112**); ***backlink***
>
> Unknown interpreted text role "c:func".

IPv4 functions took `struct sockaddr_in` structures by value, and IPv6 functions took `struct sockaddr_in6`. Now functions take a `struct sockaddr*` (note it's a pointer). It can be stack allocated.

libuv 0.10

```
struct sockaddr_in addr = uv_ip4_addr("0.0.0.0", 1234);
...
uv_tcp_bind(&server, addr)
```

libuv 1.0

```
struct sockaddr_in addr;
uv_ip4_addr("0.0.0.0", 1234, &addr)
...
uv_tcp_bind(&server, (const struct sockaddr*) &addr, 0);
```

The IPv4 and IPv6 struct creating functions (:c:func:`uv_ip4_addr` and :c:func:`uv_ip6_addr`) have also changed, make sure you check the documentation.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, **line 136**); ***backlink***
>
> Unknown interpreted text role "c:func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, **line 136**); ***backlink***

Unknown interpreted text role "c:func".

..note::

This change applies to all functions that made a distinction between IPv4 and IPv6 addresses.

Streams / UDP data receive callback API change

The streams and UDP data receive callbacks now get a pointer to a :c:type:`uv_buf_t` buffer, not a structure by value.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst, line 147); *backlink*

Unknown interpreted text role "c:type".

libuv 0.10

```
void on_read(uv_stream_t* handle,
             ssize_t nread,
             uv_buf_t buf) {
    ...
}

void recv_cb(uv_udp_t* handle,
             ssize_t nread,
             uv_buf_t buf,
             struct sockaddr* addr,
             unsigned flags) {
    ...
}
```

libuv 1.0

```
void on_read(uv_stream_t* handle,
             ssize_t nread,
             const uv_buf_t* buf) {
    ...
}

void recv_cb(uv_udp_t* handle,
             ssize_t nread,
             const uv_buf_t* buf,
             const struct sockaddr* addr,
             unsigned flags) {
    ...
}
```

Receiving handles over pipes API change

In libuv 0.10 (and earlier versions) the *uv_read2_start* function was used to start reading data on a pipe, which could also result in the reception of handles over it. The callback for such function looked like this:

```
void on_read(uv_pipe_t* pipe,
             ssize_t nread,
             uv_buf_t buf,
             uv_handle_type pending) {
    ...
}
```

In libuv 1.0, *uv_read2_start* was removed, and the user needs to check if there are pending handles using :c:func:`uv_pipe_pending_count` and :c:func:`uv_pipe_pending_type` while in the read callback:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst, line 203); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst, line 203); *backlink*

Unknown interpreted text role "c:func".

```
void on_read(uv_stream_t* handle,
             ssize_t nread,
             const uv_buf_t* buf) {
    ...
    while (uv_pipe_pending_count((uv_pipe_t*) handle) != 0) {
        pending = uv_pipe_pending_type((uv_pipe_t*) handle);
        ...
    }
    ...
}
```

Extracting the file descriptor out of a handle

While it wasn't supported by the API, users often accessed the libuv internals in order to get access to the file descriptor of a TCP handle, for example.

```
fd = handle->io_watcher.fd;
```

This is now properly exposed through the :c:func:`uv_fileno` function.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, line 231); *backlink*

Unknown interpreted text role "c:func".

uv_fs_readdir rename and API change

uv_fs_readdir returned a list of strings in the *req->ptr* field upon completion in libuv 0.10. In 1.0, this function got renamed to :c:func:`uv_fs_scandir`, since it's actually implemented using `scandir(3)`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, line 237); *backlink*

Unknown interpreted text role "c:func".

In addition, instead of allocating a full list strings, the user is able to get one result at a time by using the :c:func:`uv_fs_scandir_next` function. This function does not need to make a roundtrip to the threadpool, because libuv will keep the list of *dents* returned by `scandir(3)` around.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\[node-master][deps][uv][docs][src]migration_010_100.rst`, line 241); *backlink*

Unknown interpreted text role "c:func".