:c:type:`uv_loop_t` --- Event loop

The event loop is the central part of libuv's functionality. It takes care of polling for i/o and scheduling callbacks to be run based on different sources of events.

Data types

```
.. c:type:: uv_loop_t

   Loop data type.
```

```
.. c:enum:: uv_run_mode

   Mode used to run the loop with :c:func:`uv_run`.

   ::

        typedef enum {
            UV_RUN_DEFAULT = 0,
            UV_RUN_ONCE,
            UV_RUN_NOWAIT
        } uv_run_mode;
```

```
.. c:type:: void (*uv_walk_cb)(uv_handle_t* handle, void* arg)

   Type definition for callback passed to :c:func:`uv_walk`.
```

Public members

```
.. c:member:: void* uv_loop_t.data

   Space for user-defined arbitrary data. libuv does not use and does not
   touch this field.
```

API

Unknown directive type "c:function".

```
.. c:function:: int uv_loop_init(uv_loop_t* loop)

    Initializes the given `uv_loop_t` structure.
```

Unknown directive type "c:function".

```
.. c:function:: int uv_loop_configure(uv_loop_t* loop, uv_loop_option option, ...)

    .. versionadded:: 1.0.2

    Set additional loop options.  You should normally call this before the
    first call to :c:func:`uv_run` unless mentioned otherwise.

    Returns 0 on success or a UV_E* error code on failure.  Be prepared to
    handle UV_ENOSYS; it means the loop option is not supported by the platform.

    Supported options:

    - UV_LOOP_BLOCK_SIGNAL: Block a signal when polling for new events.  The
      second argument to :c:func:`uv_loop_configure` is the signal number.

      This operation is currently only implemented for SIGPROF signals,
      to suppress unnecessary wakeups when using a sampling profiler.
      Requesting other signals will fail with UV_EINVAL.

    - UV_METRICS_IDLE_TIME: Accumulate the amount of idle time the event loop
      spends in the event provider.

      This option is necessary to use :c:func:`uv_metrics_idle_time`.

    .. versionchanged:: 1.39.0 added the UV_METRICS_IDLE_TIME option.
```

Unknown directive type "c:function".

```
.. c:function:: int uv_loop_close(uv_loop_t* loop)

    Releases all internal loop resources. Call this function only when the loop
    has finished executing and all open handles and requests have been closed,
    or it will return UV_EBUSY. After this function returns, the user can free
    the memory allocated for the loop.
```

Unknown directive type "c:function".

```
.. c:function:: uv_loop_t* uv_default_loop(void)

    Returns the initialized default loop. It may return NULL in case of
    allocation failure.

    This function is just a convenient way for having a global loop throughout
    an application, the default loop is in no way different than the ones
    initialized with :c:func:`uv_loop_init`. As such, the default loop can (and
    should) be closed with :c:func:`uv_loop_close` so the resources associated
    with it are freed.

    .. warning::
        This function is not thread safe.
```

Unknown directive type "c:function".

```
.. c:function:: int uv_run(uv_loop_t* loop, uv_run_mode mode)

    This function runs the event loop. It will act differently depending on the
    specified mode:

    - UV_RUN_DEFAULT: Runs the event loop until there are no more active and
      referenced handles or requests. Returns non-zero if :c:func:`uv_stop`
      was called and there are still active handles or requests.  Returns
      zero in all other cases.
    - UV_RUN_ONCE: Poll for i/o once. Note that this function blocks if
      there are no pending callbacks. Returns zero when done (no active handles
      or requests left), or non-zero if more callbacks are expected (meaning
      you should run the event loop again sometime in the future).
    - UV_RUN_NOWAIT: Poll for i/o once but don't block if there are no
      pending callbacks. Returns zero if done (no active handles
      or requests left), or non-zero if more callbacks are expected (meaning
      you should run the event loop again sometime in the future).

    :c:func:`uv_run` is not reentrant. It must not be called from a callback.
```

```
.. c:function:: int uv_loop_alive(const uv_loop_t* loop)

    Returns non-zero if there are referenced active handles, active
    requests or closing handles in the loop.
```

```
.. c:function:: void uv_stop(uv_loop_t* loop)

    Stop the event loop, causing :c:func:`uv_run` to end as soon as
    possible. This will happen not sooner than the next loop iteration.
    If this function was called before blocking for i/o, the loop won't block
    for i/o on this iteration.
```

```
.. c:function:: size_t uv_loop_size(void)

    Returns the size of the `uv_loop_t` structure. Useful for FFI binding
    writers who don't want to know the structure layout.
```

```
.. c:function:: int uv_backend_fd(const uv_loop_t* loop)

    Get backend file descriptor. Only kqueue, epoll and event ports are
    supported.

    This can be used in conjunction with `uv_run(loop, UV_RUN_NOWAIT)` to
    poll in one thread and run the event loop's callbacks in another see
    test/test-embed.c for an example.

    .. note::
        Embedding a kqueue fd in another kqueue pollset doesn't work on all platforms. It's not
        an error to add the fd but it never generates events.
```

Unknown directive type "c:function".

```
.. c:function:: int uv_backend_timeout(const uv_loop_t* loop)

    Get the poll timeout. The return value is in milliseconds, or -1 for no
    timeout.
```

Unknown directive type "c:function".

```
.. c:function:: uint64_t uv_now(const uv_loop_t* loop)

    Return the current timestamp in milliseconds. The timestamp is cached at
    the start of the event loop tick, see :c:func:`uv_update_time` for details
    and rationale.

    The timestamp increases monotonically from some arbitrary point in time.
    Don't make assumptions about the starting point, you will only get
    disappointed.

    .. note::
        Use :c:func:`uv_hrtime` if you need sub-millisecond granularity.
```

Unknown directive type "c:function".

```
.. c:function:: void uv_update_time(uv_loop_t* loop)

    Update the event loop's concept of "now". Libuv caches the current time
    at the start of the event loop tick in order to reduce the number of
    time-related system calls.

    You won't normally need to call this function unless you have callbacks
    that block the event loop for longer periods of time, where "longer" is
    somewhat subjective but probably on the order of a millisecond or more.
```

Unknown directive type "c:function".

```
.. c:function:: void uv_walk(uv_loop_t* loop, uv_walk_cb walk_cb, void* arg)

    Walk the list of handles: `walk_cb` will be executed with the given `arg`.
```

Unknown directive type "c:function".

```
.. c:function:: int uv_loop_fork(uv_loop_t* loop)

    .. versionadded:: 1.12.0

    Reinitialize any kernel state necessary in the child process after
    a :man:`fork(2)` system call.

    Previously started watchers will continue to be started in the
    child process.

    It is necessary to explicitly call this function on every event
    loop created in the parent process that you plan to continue to
    use in the child, including the default loop (even if you don't
    continue to use it in the parent). This function must be called
    before calling :c:func:`uv_run` or any other API function using
    the loop in the child. Failure to do so will result in undefined
    behaviour, possibly including duplicate events delivered to both
```

```
parent and child or aborting the child process.

When possible, it is preferred to create a new loop in the child
process instead of reusing a loop created in the parent. New loops
created in the child process after the fork should not use this
function.

This function is not implemented on Windows, where it returns ``UV_ENOSYS``.

.. caution::

   This function is experimental. It may contain bugs, and is subject to
   change or removal. API and ABI stability is not guaranteed.

.. note::

    On Mac OS X, if directory FS event handles were in use in the
    parent process *for any event loop*, the child process will no
    longer be able to use the most efficient FSEvent
    implementation. Instead, uses of directory FS event handles in
    the child will fall back to the same implementation used for
    files and on other kqueue-based systems.

.. caution::

   On AIX and SunOS, FS event handles that were already started in
   the parent process at the time of forking will *not* deliver
   events in the child process; they must be closed and restarted.
   On all other platforms, they will continue to work normally
   without any further intervention.

.. caution::

   Any previous value returned from :c:func:`uv_backend_fd` is now
   invalid. That function must be called again to determine the
   correct backend file descriptor.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\(node-master)(deps)(uv)(docs)(src)loop.rst`, line 235)

Unknown directive type "c:function".

```
.. c:function:: void* uv_loop_get_data(const uv_loop_t* loop)

   Returns `loop->data`.

   .. versionadded:: 1.19.0
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\node-master\deps\uv\docs\src\(node-master)(deps)(uv)(docs)(src)loop.rst`, line 241)

Unknown directive type "c:function".

```
.. c:function:: void* uv_loop_set_data(uv_loop_t* loop, void* data)

   Sets `loop->data` to `data`.

   .. versionadded:: 1.19.0
```