Jungo Interface Layer, API Calls that will be required: all found in the auldil.h header file.
Notes:
1. DOS, Linux and Windows Interface Layer, API calls that will be required as shown below.
2. The register level will determine which functions are called as it is the only layer that can make that determination.
3. The “device” information is actually buried in the virtual_address.
4. For Jungo, all the functions below shall be blocked for use by additional devices as an active device is already using them.

```
void IO_Read_U8 ( size_t virtual_address, uint8_t  * value );
void IO_Read_U16( size_t virtual_address, uint16_t * value );

void IO_Read_Repeat_U8 ( size_t virtual_address, uint8_t  * buffer, size_t count );
void IO_Read_Repeat_U16( size_t virtual_address, uint16_t * buffer, size_t count );

void IO_Write_U8 ( size_t virtual_address, uint8_t  value );
void IO_Write_U16( size_t virtual_address, uint16_t value );

void IO_Write_Repeat_U8 ( size_t virtual_address, uint8_t  * buffer, size_t count );
void IO_Write_Repeat_U16( size_t virtual_address, uint16_t * buffer, size_t count );

int IO_Terminate( void );
int IO_Initialize( void );


TODO:  Interrupts and how to handle those criters.
```

Windows (Jungo) Required APIs (only). These APIs are called by the OS driver interface.
Notes:
1. The Open function requires: base_address, base_address_count in bytes interrupt number and returns a handle and status. The handle will be cast internally.
2. The close function requires: a device id and then returns error status. The handle will dealt with internally.
3. It will have to maintain an internal private data structure to track how often initialization is run and open/close related to each device. Also, it will need to maintian internal list of handles.
4. The function `IO_Capture()` MUST be called each time we enter the driver and then `IO_Release()` when exiting the driver. This allows the functions above to work properly as it passes the correct handle indirectly to the functions above....this is funky, but can be made to work. This also means that one and only one application can access the driver at any given time. Basically, these two functions will basically set/clear a semaphore. They could potentially sit and wait for release I suppose.

```
static struct io_device io_device[DEVICE_QTY];  //DEVICE_QTY from a shared header file.
static int io_active = 0;  //blocking semaphore, if negative, then blocking.

int IO_Capture( int device );  //maps in WDC handle and locks out additional requests
int IO_Release( int device );  //maps out WDC handle and enables requests.

int IO_Open( int    device,
             size_t base_address,
             size_t address_width_bytes,
             size_t interrupt_number );

int IO_Close( int device );

int IO_Version( unsigned long * revision );  //sys file version
```