BLOCKS

An extension to C
Not limited to ObjC, CLang, LLVM, or Apple Platforms
BSD style licensing, already ported widely

LLVM.ORG
http://clang.llvm.org/docs/LanguageExtensions.html#blocks

BLOCK SYNTAX: ^{}

lambda expression
closure
functions as first class objects
Here as an anonymous function:

```
...
dispatch_once(&tolkien, ^{   // block begins
    doThis();
    doThat(stackvar);
    // block ends
});// call to dispatch_once ends
...
```

https://en.wikipedia.org/wiki/Lambda_calculus#Lambda_calculus_and_programming_languages

LIBDISPATCH

AKA
Grand Central Dispatch

A standard C library

Callable from C, C++, ObjC...

Any language supported by gcc or CLang, (LLVM?)

Open sourced under Apache License in 2009

LIBDISPATCH

A few declarations from queue.h

```
void dispatch_sync( dispatch_queue_t queue, dispatch_block_t block);


void dispatch_async(dispatch_queue_t queue, dispatch_block_t block);


void dispatch_sync_f( dispatch_queue_t queue,
                     void *context,
                     dispatch_function_t work);



void dispatch_async_f( dispatch_queue_t queue,
                      void *context,
                      dispatch_function_t work);


dispatch_queue_t dispatch_queue_create(const char *label, dispatch_queue_attr_t attr)
  DISPATCH_QUEUE_SERIAL or DISPATCH_QUEUE_CONCURRENT

dispatch_queue_t dispatch_get_current_queue(void);
```

QUEUE VS MUTEX

```
#if USE_EngineMessageQueue

            dispatch_sync(GetEngineMessage_dispatch_queue(), ^{
                PALMouseButton( pWnd, globalMousePt, false, when, whichButton);
            }); // ends dispatch block

#else // !USE_EngineMessageQueue

            Mac_Threads_ClaimSharedMutex();
            PALMouseButton( pWnd, globalMousePt, false, when, whichButton);
            Mac_Threads_ReleaseSharedMutex();

#endif //USE_EngineMessageQueue
```

CREATING A QUEUE

```
dispatch_queue_t GA_GetScreenGCD_dispatch_queue(void)
{
    static const char* ScreenGCD_dispatch_queue_name = "com.citrix.receiver.ScreenGCD_dispatch_queue";

    static dispatch_queue_t ScreenGCD_dispatch_queue = nil; // Set nil by compiler once.
    static dispatch_once_t tolkien;                         // A magic one-shot mutex

    dispatch_once(&tolkien, ^{   // Make sure this happens only once for the entire app run.

        ScreenGCD_dispatch_queue
           = dispatch_queue_create(ScreenGCD_dispatch_queue_name, DISPATCH_QUEUE_SERIAL);

        if(!ScreenGCD_dispatch_queue) {
            // Some failure to create queue...
            SHOW("%s, could not create ScreenGCD_dispatch_queue\n", __func__);
            assert(ScreenGCD_dispatch_queue);
        }
    } );
    return ScreenGCD_dispatch_queue;
}
```

__block

The __block type declaration warns both reader and compiler that this stack variable will be modified in the block

```
/** @brief handle a change in desktop from the engine, by (re)initializing the gdc
  Either a first time initialization message or a situation where the color depth or window
  size has changed. Deletes existing objects first if required.
  Returns:  noErr, paramErr, or memory and associated os errors
*/
OSStatus GA_MacAPI_InitialiseGDC( PWND theWindowData,   //!< the window the GDC belongs to
                                  long sessionDepth)    //!< the session depth, in mac format
{
    __block OSStatus theStatus = paramErr;              //!< typed as modifiable by a block.

    if(theWindowData && theWindowData->screenGDC) {
        // parameters ok...
        HGDC theMacGDC = theWindowData->screenGDC;

        SHOWqueueAndThread;  // if macro on, displays __func__, queue name, thread name, etc.

        dispatch_sync(GA_GetScreenGCD_dispatch_queue(), ^{
            theStatus = GA_MacAPI_resetGDCframeBuffer( theMacGDC,sessionDepth,
                                                       theWindowData->fClientWidth,
                                                       theWindowData->fClientHeight);
        }); // ends block and dispatch

...
 }   // ends GA_MacAPI_InitialiseGDC
```

Simplify calls across threads

The performSelector variants do not conveniently handle parameter lists, so we end up with weird workarounds. dispatch with a block cleans up code significantly

```
void MakePhoneCall(const UTF8Char *phoneCallNumber, int uniqueId, UInt16 transactionId)
{
    SessionViewController* svc = [[[UIApplication sharedApplication] delegate sessionViewController]];

#if USE_MainThreadQueue

    dispatch_async(dispatch_get_main_queue(),^{
        [svc makePhoneCallTo:phoneCallNumber withUniqueId:uniqueId
                                       withTransactionId:transactionId];
    });

#else // !USE_MainThreadQueue

    NSString *phoneNumber = [NSString stringWithUTF8String:(char *)phoneCallNumber];
    NSDictionary *params = [NSDictionary dictionaryWithObjectsAndKeys:
                            phoneNumber , @"phonenumber",
                            [NSNumber numberWithInt:uniqueId], @"uniqueid",
                            [NSNumber numberWithShort:transactionId], @"transactionid",nil];
       [svc performSelectorOnMainThread:@selector(makePhoneCall:) withObject:params waitUntilDone:NO];

#endif // !USE_MainThreadQueue

}
```