**UNDERSTANDING .DMP FILES**

These files contain a copy of the computer’s memory at the time of the crash. They can be used to help diagnose and identify root cause of a crash. Crashes are caused by unhandled exceptions in code There are several types and categories of DMP Files but can be broken down into 2 main categories:

**Kernel mode** - Refers to the state of the CPU, used when processing instructions that require the kernel-level services. Executing code has access to all memory addresses. Crashes here will result in a kernel dump, all system activity halts to process the crash -and the dreaded blue screen error message while the crash is processed.

**User mode** - Refers to the state of the CPU, used when processing normal work. Executing code has no direct access to hardware or memory (they must make a request to the kernel-level services). Crashes here have no effect on the kernel, so the OS should maintain operability and affect only the specific process.

To make it easy:

An application crash = user mode dumps

A system crash/BSOD = kernel mode dumps

When a crash does occur in either mode, a dump file is created However, we will only be focusing on User Mode dump files because we only troubleshoot the individual eDOCS processes such as DM.EXE or DOCSFUSION.EXE

**User dump -** The type of dump can be a Mini dump or Full dump. Created by Debug tools such as DebugDiag. This is created when an irrecoverable error occurs (such as an unhandled exception) within a running process, causing that process to crash.

Note:

DebugDiag creates a “User Mini Dump with Full Memory”. This includes all accessible memory in the process which can make for a large file. The dmp file type is not configurable in the interface and is set to only capture second chance exceptions by default. However, you can configure the type of exceptions you want to capture

**WHAT ARE EXCEPTIONS?**

Exceptions are a type of error that occurs during the execution of an application. However, are not the same as an error. Errors are typically problems that are not expected. Whereas, exceptions can be expected to happen within the application’s code for various reasons. This means exceptions are not innately bad- it just matters if they are handled or not. Exceptions are related to application whereas errors are related to environment in which application is running.

Since exceptions can expected to happen you can break them down into 2 types:

Handled Exceptions (First Chance)

Unhandled Exceptions (Second Chance) = (Crash)

Applications use exception handling logic to explicitly handle the exceptions when they happen. Exceptions can occur for a wide variety of reasons. From the infamous NullReferenceException to a database query timeout.

*Ex. Unhandled Exception*

static void Main(string[] args)

{

var a = 3;

var b = 0;

//dividing a number by 0 will throw a "System.DivideByZeroException:

var result = a / b;

}

This causes the app to crash

*An unhandled exception of type 'System.DivideByZeroException' occurred in Exceptions.exe Attempted to divide by zero.*

*Ex. Handled Exception*

static void Main(string[] args)

{

var a = 3;

var b = 0;

try

{

//dividing a number by 0 will throw a "System.DivideByZeroException:

var c= a / b;

Console.WriteLine(c);

}

catch ( DivideByZeroException ex) //This handles the exception -no crash

{

Console.WriteLine(ex.Message);

Console.WriteLine("Application still running ");

}

}

**WHAT ARE SYMBOLS?**

When you are debugging an application that has crashed, the debugger attempts to show you the functions on the stack that led up to the crash. Without symbol files, the debugger cannot resolve the function names, their parameters, or any local variables the are stored on the stack.

Typically, symbol files might contain:

- Global variables

- Local variables

- Function names

- Source-line numbers

Each of these items is called, individually, a symbol. For example, a single symbol file Myprogram.pdb might contain several hundred symbols, including global variables and function names and hundreds of local variables.

When debugging, you must make sure that the debugger can access the symbol files that are associated with the target you are debugging. Both live debugging and debugging crash dump files require symbols. You must obtain the proper symbols for the code that you wish to debug and load these symbols into the debugger.

**STACKS..HEAPS..MEMORY LEAKS**

The stack is a region of your computer's memory that stores temporary variables created by each function. The stack is a "LIFO" (last in, first out) data structure, that is managed and optimized by the CPU. Every time a function declares a new variable, it is "pushed" onto the stack. Then every time a function exits, all of the variables pushed onto the stack by that function, are freed (they are deleted). Once a stack variable is freed, that region of memory becomes available for other stack variables.

The heap is a region of your computer's memory that is not managed automatically for you and is not as tightly managed by the CPU. It is a more free-floating region of memory. If you have allocated memory on the heap, you are responsible for deallocate that memory once you don't need it anymore. If you fail to do this, your program will have what is known as a memory leak.