[Execution Context of Threads](http://stackoverflow.com/questions/1951924/executioncontext-of-threads)

The details of ExecutionContext are very obscure, buried deep inside features like .NET Remoting and WCF. What is part of it is:

* HostExecutionContext
* IllogicalCallContext, a repository of thread specific data used by Remoting
* LogicalContext, as above
* SecurityContext
* SynchronizationContext

CultureInfo is not part of it, which can be a considerable problem if you change your main thread's default culture. There is no good way to ensure other threads run with that culture unless you explicitly write the code to switch them. That's not always practical, given that .NET is apt to run async callbacks on threadpool threads. They will be initialized to the system default culture.

Edit: this problem got fixed in .NET 4.5 with the CultureInfo.DefaultThreadCurrentCulture property.

ExcecutionContext.\*SuppressFlow\* suppresses the flow of the execution context across asynchronous threads.

The **ExecutionContext**, are implicitly passed from parent thread to the child one, provides information relevant to a logical thread of execution: security context, call context and synchronization context. If that information is not imperative, the omission of the execution context optimize a little the performance of a multithreading application.

ExecutionContext.\*RestoreFlow\* restores the passage of the execution context between threads.

Finally

**Q**: In the following code what exactly gets suppressed??

**A**: Exactly are suppressed the passage of the following information: security context, call context and synchronization context; between the newly created threads. Why that was do? -To optimize the creation and work of th.Length created threads: less supplementary information passed between threads - quicker this threads interact between them.