# ManagedCHESS Integration in Visual Studio Team Test

Chess Team and Peli Halleux

ManagedCHESS (MC) is a concurrency checker that detects races, deadlocks and other concurrency bugs in managed code without any source code change. The goal of this integration is to provide a simple way to allow Team Test users to use ManagedCHESS via the Unit Test feature.

## ManagedChess in a nutshell

* **Iterative:** MC takes a .NET method (which uses multi-threading) and runs it iteratively, exploring (in the limit) all possible thread schedules (using efficient search space pruning algorithms).
* **Repro:** When it finds a failure, MC emits a ‘schedule’ file that can be used to reproduce the thread schedule that caused the failure.
* **Extended Reflection:** MC uses Extended Reflection (rewriting framework built for Pex) to hook the runtime behavior. In that sense, it works on any MSIL code and does not require source changes.

## Integration: HostType

MC can be seen as a Unit Test host type, taking any Team Test Unit Test and running it iteratively, exploring a different thread schedule in each iteration. This integration approach makes it extremely easy for the user to use MC:

[TestMethod]  
**[HostType(“Chess”)]**public void MyConcurrentTest() {  
 ...  
}

The MC host would launch the MC console application on each test and configure it to run that particular test. The console output becomes the test output, the MC console exit code is used to determine if the test passed.

When MC finds an issue, it has to be surfaced to the user as a new unit test. MC generates a unit test in the test output that the user can copy/paste in the project to repro the issue. The ‘schedule’ is serialized as a string and passed to a new attribute [ChessSchedule]. The MC host would detect this schedule and automatically update switch to run in repro mode:

**const string Schedule = @“xx usduxu xixix”;**

[TestMethod]  
[HostType(“Chess”)]  
**[ChessSchedule(Schedule)]**public void MyConcurrentTestRepro() {  
 this.MyConcurrentTest();  
}

## Methodology

* Unit Test and Chess RunTest methodology
* **TODO**

## Default Behavior and Attributes

* [HostType(“Chess”)]
* [ChessSchedule(Schedule)]
* **TODO**

## Workitems

1. **VS Integration (Tom and Peli)**
   1. Finish HostType implementation that supports mstest unit tests without hacks
   2. Special handling of internal test execution paths: mstest does not expose directly the method delegate under test so we need to be smart in turning on monitoring only when we enter user code to avoid tracing mstest infrastructure. **There’s some magic to do here and it’s critical.**
   3. Prettify MC output to be displayed in details view
   4. Error codes
      1. Red: real error
      2. Yellow: don’t know
      3. Green: no error
   5. Generate the repro test case when found (find something that is easy/looks good)
   6. Good debugging experience on repro (hide MC and ER stack frames)
   7. Add support for ChessSchedule in host
   8. Create ManagedChess installer with common ER merge module
2. **MC wrappers for System.Threading (Shaz)**
   1. Review wrappers and document their status
   2. Update wrappers and test more completely
      1. Completeness
      2. Checking pre-conditions
   3. Plans for other APIs other than System.Threading ???
3. **ManagedCHESS**
   1. Parameterize Exit
4. **CHESS Engine (Madan,Sebastian)**
   1. Parameterize Exit
   2. Default behavior
      1. /csb:1 /volatile
   3. Disable printing on error messages
   4. Deadlock
   5. Livelock
   6. Race detection (post-PDC)
   7. Best-first search (post-PDC)
   8. Relaxed memory models (post-PDC)

## Future integration

Integrate visualization tools, etc… This is not critical to get it out of the door.