MetaAutomation Sample1

Tutorial

Contents

[Summary 2](#_Toc468974188)

[Tutorial Objectives 2](#_Toc468974189)

[Tutorial Steps 2](#_Toc468974190)

[1. Installing necessary software 2](#_Toc468974191)

[2. Creating Files and Classes 3](#_Toc468974192)

[Creating a C# Implementation Class 3](#_Toc468974193)

[3. Implementing a Simple Check 4](#_Toc468974194)

[4. Connecting the pieces 6](#_Toc468974195)

[Create an Initial Artifact File 6](#_Toc468974196)

[Point the CheckMap to the Artifact File 7](#_Toc468974197)

[5. Executing the Check for the First Time 9](#_Toc468974198)

[6. Configuring the Check 9](#_Toc468974199)

[7. Running the Check 11](#_Toc468974200)

[8. Modifying the Check 11](#_Toc468974201)

[9. Looking Ahead 12](#_Toc468974202)

# Summary

This exercise accompanies the MetaAutomation Sample1, and shows how one can begin implementing checks with the libraries of Sample1.

## Tutorial Objectives

Students who complete the tutorial will learn these objectives, using the provided sample library:

1. How to implement an atomic check that self-documents at runtime according to the Hierarchical Steps pattern of MetaAutomation
2. How to configure the check
3. How to run the check and view results
4. How to modify the check, run and view the results

# Tutorial Steps

## Installing necessary software

Please install and run the MetaAutomation Sample 1 first. Refer to the installation instructions that come with Sample 1, and run both sample checks 1 and 2 included with Sample 1.

Install the code editor extension to Visual Studio:

1. Select Tools -> Extensions and Updates…
2. In the “Extensions and Updates” tools, at left, pick Online -> Visual Studio Gallery
3. At upper right, for “Search Visual Studio Gallery,” type “metaautomation”
4. Select, download and install “Check Step Editor”
5. Restart Visual Studio

The following instructions assume that the Sample 1 solution is open in Visual Studio 2015. The steps refer to working with the solution. The colored text examples shown below assume the default text editor coloration for Visual Studio.

## Creating Files and Classes

### Creating a C# Implementation Class

In the CheckMethods project of solution MetaAutomationSample1, create a new class, called e.g. “AtomicChecks.”

In the source file for the new class, make these changes:

1. Make the class public
2. Import the single-process MetaAutomation client library, with the statement

using MetaAutomationClientSpLibrary;

1. Create a public instance method for the class, e.g.

public void AtomicCheck\_Test()

1. Add a CheckMethodAttribute to the class, with the method name and a unique GUID

[CheckMethod(CheckMethodName = "AtomicCheck\_Test", CheckMethodGuid = "8821B991-D2D6-4FEE-8828-0C0A33F8BC26")]

public void AtomicCheck\_Test()

To make a new, unique GUID, use the Tools menu and select “Create GUI.” Select the “Registry Format” radio button, click the “New GUID” button, then the “Copy” button. Paste the GUID into the quoted string for the CheckMethodGuid initializer for the attribute, and remove the curly braces.

1. Add a try/catch block with the error reporting handler
2. Add your dependencies and code to implement a simple check. Some placeholder code is shown here

[CheckMethod(CheckMethodName = "AtomicCheck\_Test", CheckMethodGuid = "8821B991-D2D6-4FEE-8828-0C0A33F8BC26")]

public void AtomicCheck\_Test()

{

try

{

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

}

catch (System.Exception ex)

{

Check.ReportFailureData(ex);

}

}

## Implementing a Simple Check

To follow the Atomic Check pattern requires also following the Hierarchical Steps pattern. All steps of the check are self-documenting at runtime, from the high-level business-facing steps that are meaningful to anybody on the team who knows and understands the software to the low-level technology-facing steps that drive and measure the software system.

To do this, use the “Check Step Editor” extension:

1. In the text editor for the C# source code, place the cursor or select a word on a line of code in the check. For example, this could be the first of the five Sleep statements.
2. Select the Edit menu -> Add Check Step (or, Alt-E, E). The code should look like this:

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

1. Do the same for the other four lines of code, creating this:

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

1. Select all the lines of code inside the try block and Edit menu -> Add Check Step (or, Alt-E, E). All selected code is now placed in a larger-scope step, like this:

Check.Step("xxxx.", delegate

{

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("xxxx.", delegate

{

System.Threading.Thread.Sleep(200);

});

});

1. To create high-quality data on driving the product, the names of the check steps must be hard-coded and unique, so after editing and creating those names the code might look like this:

Check.Step("Stepped sleeping.", delegate

{

Check.Step("First sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Second sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Third sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Fourth sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Fifth sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

});

1. If you would like to clean the check steps and start again, select all lines of code for which the check step statements are to be cleaned up and use the Edit Menu -> Remove Check Steps (or, Alt-E, M). In this simplistic example, this will now look like:

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

System.Threading.Thread.Sleep(200);

1. Type Ctrl-Z to undo the edit of step 6. All edits done through the extension are included in the undo-redo queue of the editor. The entire method should now look like this:

[CheckMethod(CheckMethodName = "AtomicCheck\_Test", CheckMethodGuid = "8821B991-D2D6-4FEE-8828-0C0A33F8BC26")]

public void AtomicCheck\_Test()

{

try

{

Check.Step("Stepped sleeping.", delegate

{

Check.Step("First sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Second sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Third sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Fourth sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

Check.Step("Fifth sleep.", delegate

{

System.Threading.Thread.Sleep(200);

});

});

}

catch (System.Exception ex)

{

Check.ReportFailureData(ex);

}

}

## Connecting the pieces

### Create an Initial Artifact File

The artifact files for a check serve two purposes:

1. They are a record of what happened when the check was last run
2. They are a guide for running the check next time

This will become more clear through this tutorial and the example checks that are part of Sample 1. The artifact files become still more important for more sophisticated checks that cross processes and tiers, as with the MetaAutomation samples 2 and 3.

Follow these steps to create an initial artifact file for the check of the tutorial:

1. From the Solution Explorer, right-click over the solution and select “Open Folder in File Explorer.”
2. Click or double-click to open the folder “Artifacts.” Note that the file CheckMap.xml is here; this will be important soon.
3. Create a folder called e.g. “AtomicCheck\_Test” as a child of the Artifacts folder.
4. Copy the file “CheckRunArtifact\_Tutorial.xml” into the new folder.
5. Rename the new file “CheckRunArtifact\_Original.xml”
6. Open the new file in Visual Studio to edit it

This will require editing some XML, but if this is outside of your experience note that it’s just another kind of text file so it shouldn’t be hard; just follow the instructions carefully:

1. Note that there are two lines in the file with zero-length strings, looking like this:

<DataElement Name="CheckMethodName" Value="" />

<DataElement Name="CheckMethodGuid" Value="" />

1. From the C# source code of your check implementation, copy the quoted strings for CheckMethodName and CheckMethodGuid, respectively, into the file. It might look like this:

<DataElement Name="CheckMethodName" Value="AtomicCheck\_Test" />

<DataElement Name="CheckMethodGuid" Value="8821B991-D2D6-4FEE-8828-0C0A33F8BC26" />

1. If you’ve done this correctly, the artifact file “CheckRunArtifact\_Original.xml” will look like this:

<?xml version="1.0" encoding="utf-8"?>

<CheckRunArtifact>

<CheckRunData>

<DataElement Name="CheckJobSpecGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckJobRunGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckRunGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckMethodName" Value="AtomicCheck\_Test" />

<DataElement Name="CheckMethodGuid" Value="8821B991-D2D6-4FEE-8828-0C0A33F8BC26" />

<DataElement Name="CheckBeginTime" Value="2015-12-15T22:18:31.5551054Z" />

<DataElement Name="CheckEndTime" Value="2015-12-15T22:18:31.7020476Z" />

</CheckRunData>

<CheckCustomData />

<CheckFailData />

<CompleteCheckStepInfo/>

</CheckRunArtifact>

### Point the CheckMap to the Artifact File

The check map file “CheckMap.xml” indicates which checks to run and points to the artifact files required to run the check or checks. The MetaAutomation libraries update the check map to point to new artifact files as they are created.

In the Solution Explorer window, under solution MetaAutomationSample1 there is a folder called “Artifacts,” and in that folder is CheckMap.xml. Double-click the file to edit it.

On downloading and installing MetaAutomation sample 1, the check map might look like this:

<?xml version="1.0" encoding="utf-8"?>

<Checks>

<Check>

<DataElement Name="DirectoryName" Value="Example\_1\_SimpleOneTierCheck" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Example 1: Simple one-tier check" />

</Check>

<!--Check>

<DataElement Name="DirectoryName" Value="Example\_2\_CheckOfWebPageBuiltToFailRandomly" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Example 2: Check of web page built to fail randomly" />

</Check-->

</Checks>

The following instructions assume knowledge of XML, but there are some graphics following to show what is intended:

1. Copy the first Check element (as it appears on lines 3-7 above)
2. Comment out the Check element that was just copied, so it may look like this

<?xml version="1.0" encoding="utf-8"?>

<Checks>

<Check>

<DataElement Name="DirectoryName" Value="Example\_1\_SimpleOneTierCheck" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Example 1: Simple one-tier check" />

</Check>

<!--Check>

<DataElement Name="DirectoryName" Value="Example\_1\_SimpleOneTierCheck" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Example 1: Simple one-tier check" />

</Check-->

<!--Check>

<DataElement Name="DirectoryName" Value="Example\_2\_CheckOfWebPageBuiltToFailRandomly" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Example 2: Check of web page built to fail randomly" />

</Check-->

</Checks>

1. In the DataElement element with attribute Name=”DirectoryName” edit the Value string to be the new directory created for the artifact files of the new check.
2. In the DataElement element with attribute Name=”CurrentCheckRunArtifact” edit the Value string to be the name of the current (original) artifact file of the new check. Be sure to include the “.xml” file extension.
3. (optional) In the DataElement element with attribute Name=”Comment” edit the Value string to be what we’re doing here: “Sample 1 Tutorial.”
4. The check map should look like this:

<?xml version="1.0" encoding="utf-8"?>

<Checks>

<Check>

<DataElement Name="DirectoryName" Value="AtomicCheck\_Test" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Sample 1 tutorial" />

</Check>

<!--Check>

<DataElement Name="DirectoryName" Value="Example\_1\_SimpleOneTierCheck" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Example 1: Simple one-tier check" />

</Check-->

<!--Check>

<DataElement Name="DirectoryName" Value="Example\_2\_CheckOfWebPageBuiltToFailRandomly" />

<DataElement Name="CurrentCheckRunArtifact" Value="CheckRunArtifact\_Original.xml" />

<DataElement Name="Comment" Value="Example 2: Check of web page built to fail randomly" />

</Check-->

</Checks>

## Executing the Check for the First Time

Check that the build configuration is “Debug.”

Select Build -> Rebuild solution, to ensure that the entire solution is built. This is necessary because the library/assembly with your check code, CheckMethods.dll, is loaded at runtime, just as with the xUnit frameworks, so Visual Studio might not recognize that it needs to be rebuilt to include the code edits you made above.

Hit F5 to run through your check.

If the run does not complete in 10 seconds or so, and the command-line environment that pops up seems to be full of text, there is a configuration error and some problem-solving ahead. Please double-check the prior steps, read the error, ask a nearby colleague, or if all else fails, contact me through the web site MetaAutomation.net and send a complete description of what you did and what happened and a screen shot of the error or copied text.

It's convenient to comment out parts of the check map file to run other parts, to keep the commented-out sections as reference. But, all the un-commented Check elements in the check map will run checks sequentially for MetaAutomation Sample 1 download.

## Configuring the Check

On a successful run of the check, the check map is updated to point to the new artifact file, and the new artifact file is in the same directory as the original, with name “CheckRunArtifact\_<GUID>.xml.” Double-click the new artifact file to open and edit it:

<?xml version="1.0" encoding="utf-8"?>

<CheckRunArtifact>

<CheckRunData>

<DataElement Name="CheckJobSpecGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckJobRunGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckRunGuid" Value="824956a3-e91f-4bbd-ac7f-46f12dac0947" />

<DataElement Name="CheckMethodName" Value="AtomicCheck\_Test" />

<DataElement Name="CheckMethodGuid" Value="8821B991-D2D6-4FEE-8828-0C0A33F8BC26" />

<DataElement Name="CheckBeginTime" Value="2016-11-29T21:49:30.0265599Z" />

<DataElement Name="CheckEndTime" Value="2016-11-29T21:49:31.0485340Z" />

</CheckRunData>

<CheckCustomData />

<CheckFailData />

<CompleteCheckStepInfo>

<CheckStep Name="Method AtomicCheck\_Test" msTimeLimit="30000" msTimeElapsed="1017" Value="Pass">

<CheckStep Name="Stepped sleeping." msTimeLimit="30000" msTimeElapsed="1014" Value="Pass">

<CheckStep Name="First sleep." msTimeLimit="30000" msTimeElapsed="201" Value="Pass" />

<CheckStep Name="Second sleep." msTimeLimit="30000" msTimeElapsed="201" Value="Pass" />

<CheckStep Name="Third sleep." msTimeLimit="30000" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Fourth sleep." msTimeLimit="30000" msTimeElapsed="201" Value="Pass" />

<CheckStep Name="Fifth sleep." msTimeLimit="30000" msTimeElapsed="200" Value="Pass" />

</CheckStep>

</CheckStep>

</CompleteCheckStepInfo>

</CheckRunArtifact>

Note that the check step hierarchy documented itself at runtime. Each check step has an attribute for time elapsed; note that for a parent step, time elapsed includes execution of all children. For example the check step with name “Method AtomicCheck\_Test” was automatically generated for the entry point method for the check, and shows the entire elapsed time for the check run.

The time limit for each step defaulted to 30,000 milliseconds, i.e., 30 seconds. This default is determined by MetaAutomation library code, but the time limits / timeouts are configured in the artifact file, now that we have a complete check run with all of the steps. If the timeouts are configured to be a little more realistic (given that the tutorial code given is trivial) it might look like this, with the msTimeLimit attributes edited:

<?xml version="1.0" encoding="utf-8"?>

<CheckRunArtifact>

<CheckRunData>

<DataElement Name="CheckJobSpecGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckJobRunGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckRunGuid" Value="824956a3-e91f-4bbd-ac7f-46f12dac0947" />

<DataElement Name="CheckMethodName" Value="AtomicCheck\_Test" />

<DataElement Name="CheckMethodGuid" Value="8821B991-D2D6-4FEE-8828-0C0A33F8BC26" />

<DataElement Name="CheckBeginTime" Value="2016-11-29T21:49:30.0265599Z" />

<DataElement Name="CheckEndTime" Value="2016-11-29T21:49:31.0485340Z" />

</CheckRunData>

<CheckCustomData />

<CheckFailData />

<CompleteCheckStepInfo>

<CheckStep Name="Method AtomicCheck\_Test" msTimeLimit="2000" msTimeElapsed="1017" Value="Pass">

<CheckStep Name="Stepped sleeping." msTimeLimit="1800" msTimeElapsed="1014" Value="Pass">

<CheckStep Name="First sleep." msTimeLimit="300" msTimeElapsed="201" Value="Pass" />

<CheckStep Name="Second sleep." msTimeLimit="300" msTimeElapsed="201" Value="Pass" />

<CheckStep Name="Third sleep." msTimeLimit="300" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Fourth sleep." msTimeLimit="000" msTimeElapsed="201" Value="Pass" />

<CheckStep Name="Fifth sleep." msTimeLimit="300" msTimeElapsed="200" Value="Pass" />

</CheckStep>

</CheckStep>

</CompleteCheckStepInfo>

</CheckRunArtifact>

## Running the Check

Save any msTimeLimit edits to the artifact file, run the check again, and open the resulting artifact:

<?xml version="1.0" encoding="utf-8"?>

<CheckRunArtifact>

<CheckRunData>

<DataElement Name="CheckJobSpecGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckJobRunGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckRunGuid" Value="f6050b9c-ac52-4a70-8ff2-6c1f3e35ba98" />

<DataElement Name="CheckMethodName" Value="AtomicCheck\_Test" />

<DataElement Name="CheckMethodGuid" Value="8821B991-D2D6-4FEE-8828-0C0A33F8BC26" />

<DataElement Name="CheckBeginTime" Value="2016-11-29T22:06:01.5840726Z" />

<DataElement Name="CheckEndTime" Value="2016-11-29T22:06:02.5976390Z" />

</CheckRunData>

<CheckCustomData />

<CheckFailData />

<CompleteCheckStepInfo>

<CheckStep Name="Method AtomicCheck\_Test" msTimeLimit="2000" msTimeElapsed="1010" Value="Pass">

<CheckStep Name="Stepped sleeping." msTimeLimit="1800" msTimeElapsed="1007" Value="Pass">

<CheckStep Name="First sleep." msTimeLimit="300" msTimeElapsed="201" Value="Pass" />

<CheckStep Name="Second sleep." msTimeLimit="300" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Third sleep." msTimeLimit="300" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Fourth sleep." msTimeLimit="300" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Fifth sleep." msTimeLimit="300" msTimeElapsed="200" Value="Pass" />

</CheckStep>

</CheckStep>

</CompleteCheckStepInfo>

</CheckRunArtifact>

Note that since the check steps, as determined by the C# code, have not changed, the msTimeLimit attributes for timeouts for each step have not changed.

## Modifying the Check

Modify the check method:

1. Select the C# code for the “First sleep” and “second sleep” steps. It’s not necessary to select the entire lines, just to have the selection begin somewhere on the Check.Step line and end on the line with the closing semicolon, and cross those two steps.
2. Edit -> Add Check Step (or, Alt-E, E)
3. Name this new step “foo.”
4. Do the same for the 3rd, 4th and 5th sleeps, so they’re grouped as well in a step
5. Name this new step “bar.”
6. Build -> Rebuild Solution
7. F5 to run it again in debug
8. Open the new artifact

<?xml version="1.0" encoding="utf-8"?>

<CheckRunArtifact>

<CheckRunData>

<DataElement Name="CheckJobSpecGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckJobRunGuid" Value="00000000-0000-0000-0000-000000000000" />

<DataElement Name="CheckRunGuid" Value="bbbb5554-b754-40f9-980b-305e5dfb6d46" />

<DataElement Name="CheckMethodName" Value="AtomicCheck\_Test" />

<DataElement Name="CheckMethodGuid" Value="8821B991-D2D6-4FEE-8828-0C0A33F8BC26" />

<DataElement Name="CheckBeginTime" Value="2016-11-29T22:19:29.1846082Z" />

<DataElement Name="CheckEndTime" Value="2016-11-29T22:19:30.2016807Z" />

</CheckRunData>

<CheckCustomData />

<CheckFailData />

<CompleteCheckStepInfo>

<CheckStep Name="Method AtomicCheck\_Test" msTimeLimit="2000" msTimeElapsed="1010" Value="Pass">

<CheckStep Name="Stepped sleeping." msTimeLimit="1800" msTimeElapsed="1007" Value="Pass">

<CheckStep Name="foo." msTimeLimit="30000" msTimeElapsed="404" Value="Pass">

<CheckStep Name="First sleep." msTimeLimit="30000" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Second sleep." msTimeLimit="30000" msTimeElapsed="200" Value="Pass" />

</CheckStep>

<CheckStep Name="bar." msTimeLimit="30000" msTimeElapsed="602" Value="Pass">

<CheckStep Name="Third sleep." msTimeLimit="30000" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Fourth sleep." msTimeLimit="30000" msTimeElapsed="200" Value="Pass" />

<CheckStep Name="Fifth sleep." msTimeLimit="30000" msTimeElapsed="200" Value="Pass" />

</CheckStep>

</CheckStep>

</CheckStep>

</CompleteCheckStepInfo>

</CheckRunArtifact>

Please notice two things:

1. The time limits for the beginning steps, that are recognized as existing steps, are preserved. If the steps don’t actually change, the time limits are always preserved.
2. The newly recognized steps, including steps that are now nested as children in “foo.” And “bar,” have the default time limits. To update these, the artifact file must be edited in place, just as above.

## Looking Ahead

Example check 2, a check of a web page built to fail randomly, comes with the MetaAutomation download Sample 1 and shows how failures and timeouts get much more interesting with a more realistic (although still very simple) check. Failures cascade up through parent steps, steps that are prevented from executing by earlier failures are marked “Blocked,” and when a parent step times out when a child step is running, both steps are noted in the thrown exception and documented in the artifact from the check run.

The MetaAutomation Sample 2 download does everything the Sample 1 download does *plus* it parallelizes checks in multiple processes, and shows how a single check can run across processes and leave a single coherent, valid XML document after the check run, which comes in handy for product instrumentation or OS service management.

The MetaAutomation Sample 3 download does everything the Sample 2 download does plus it parallelizes checks across tiers/machines and can run a single check across any number of machines. This enables much more efficient automated checking for products that cross tiers for Internet of Things.