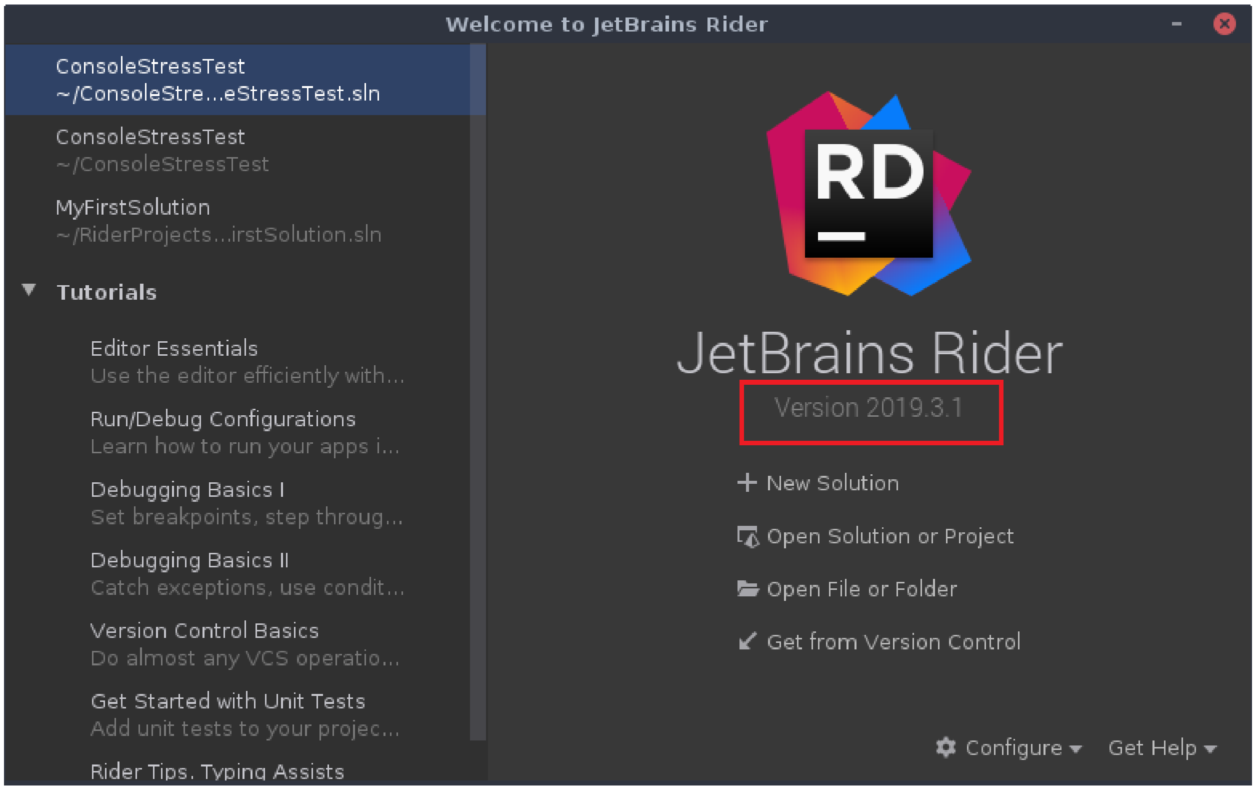
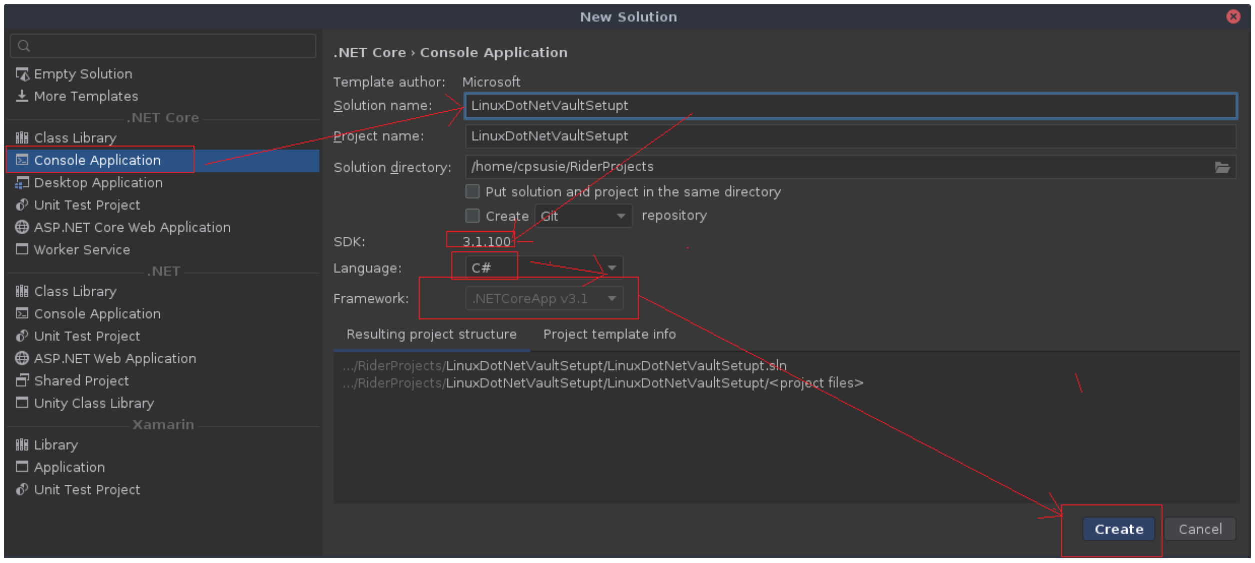
DotNetVault Quick Start Guide – JetBrains Rider 2019.3.1 (Tested on Amazon Linux)

1- Make sure you have JetBrains Rider 2019.3.1 or later installed and have selected .NET Core 3.1+ as your framework


Welcome to JetBrains Rider
ConsoleStressTest
~/ConsoleStre...eStressTest.sln
ConsoleStressTest
~/ConsoleStressTest
MyFirstSolution
~/RiderProjects...irstSolution.sln
Tutorials
Editor Essentials
Use the editor efficiently with...
Run/Debug Configurations
Learn how to run your apps i...
Debugging Basics I
Set breakpoints, step throug...
Debugging Basics II
Catch exceptions, use condit...
Version Control Basics
Do almost any VCS operatio...
Get Started with Unit Tests
Add unit tests to your projec...
Rider Tips, Typing Assists
RD
JetBrains Rider
Version 2019.3.1
New Solution
Open Solution or Project
Open File or Folder
Get from Version Control
Configure
Get Help

2- Make sure Roslyn Analyzers are enabled. See: https://www.jetbrains.com/help/rider/Settings_Roslyn_Analyzers.html

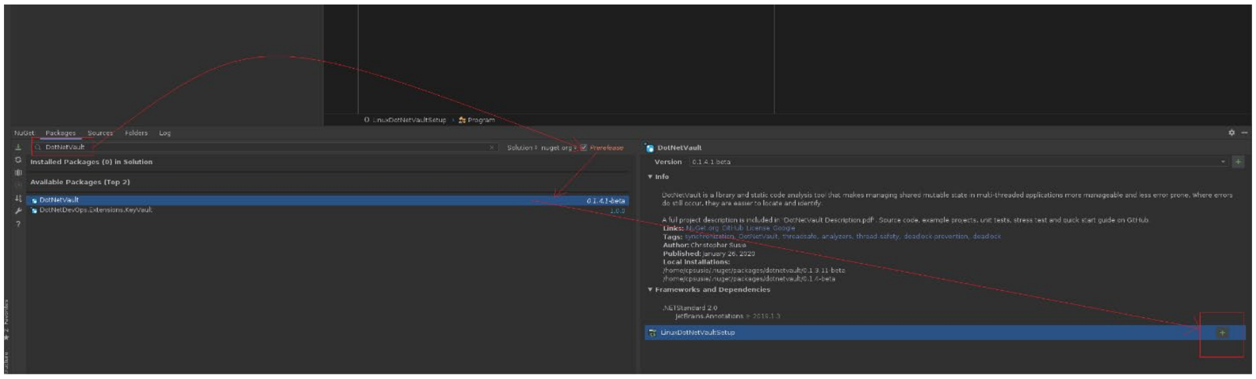
3- Create a new .NET Core Console project as shown:



Template author: Microsoft
Solution name: LinuxDotNetVaultSetup
Project name: LinuxDotNetVaultSetup
Solution directory: /home/cpsusie/RiderProjects
Put solution and project in the same directory
Create Git repository
SDK: 3.1.100
Language: C#
Framework: .NETCoreApp v3.1

```
.../RiderProjects/LinuxDotNetVaultSetup/LinuxDotNetVaultSetup.sln
.../RiderProjects/LinuxDotNetVaultSetup/LinuxDotNetVaultSetup/<project files>
```

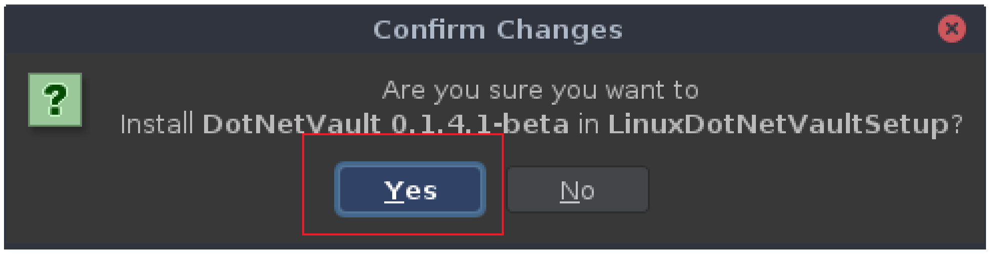
4- Right click on your project and chose “Manage NuGet packages”.
5- As shown enter “DotNetVault” into the search bar in the NuGet Panel, make sure “Prerelease” is checked, select the latest version (0.1.4.1-beta or later) of DotNetVault, then click the plus button. Choose “Yes” in the confirmation dialog.


LinuxDotNetVaultSetup › Program
NuGet
Packages
Sources
Folders
Log
DotNetVault
Solution
nuget.org
Prerelease
Installed Packages (0) in Solution
Available Packages (Top 2)
DotNetVault 0.1.4.1-beta
DotNetDevOps.Extensions.KeyVault 1.0.0
DotNetVault
Version 0.1.4.1-beta
Info
Frameworks and Dependencies
LinuxDotNetVaultSetup

DotNetVault is a library and static code analysis tool that makes managing shared mutable state in multi-threaded applications more manageable and less error prone. Where errors do still occur, they are easier to locate and identify.

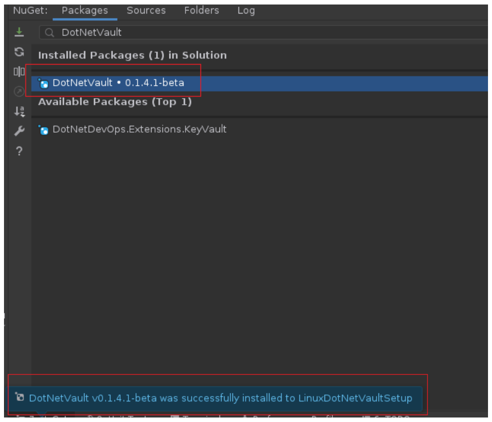
A full project description is included in 'DotNetVault Description.pdf'. Source code, example projects, unit tests, stress test and quick start guide on GitHub.
Links: NuGet.org GitHub License Google
Tags: synchronization, DotNetVault, threadsafe, analyzers, thread-safety, deadlock-prevention, deadlock
Author: Christopher Susie
Published: January 26, 2020
Local Installations:
/home/cpsusie/.nuget/packages/dotnetvault/0.1.3.11-beta
/home/cpsusie/.nuget/packages/dotnetvault/0.1.4-beta

.NETStandard 2.0
JetBrains.Annotations ≥ 2019.1.3



Are you sure you want to
Install **DotNetVault 0.1.4.1-beta** in **LinuxDotNetVaultSetup**?

6- If installation succeeded, should see something like the below:


NuGet:
Packages
Sources
Folders
Log
DotNetVault
Installed Packages (1) in Solution
DotNetVault • 0.1.4.1-beta
Available Packages (Top 1)
DotNetDevOps.Extensions.KeyVault

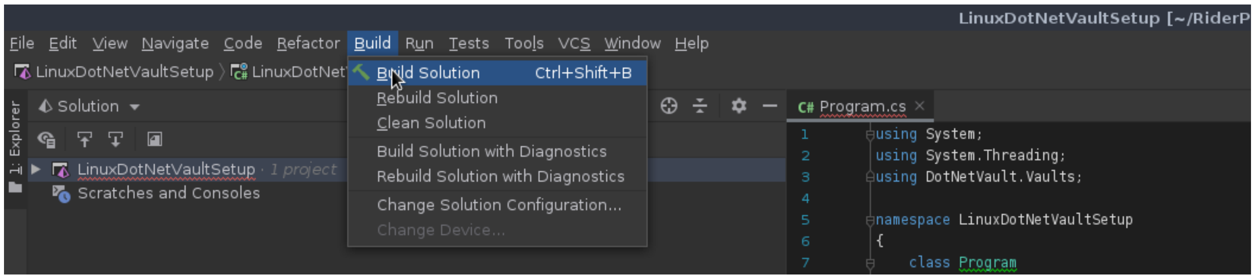
DotNetVault v0.1.4.1-beta was successfully installed to LinuxDotNetVaultSetup

7-

a. To ensure that the static analyzer installed correctly, delete the “Hello World” program entered in by default and replace it with the following code:

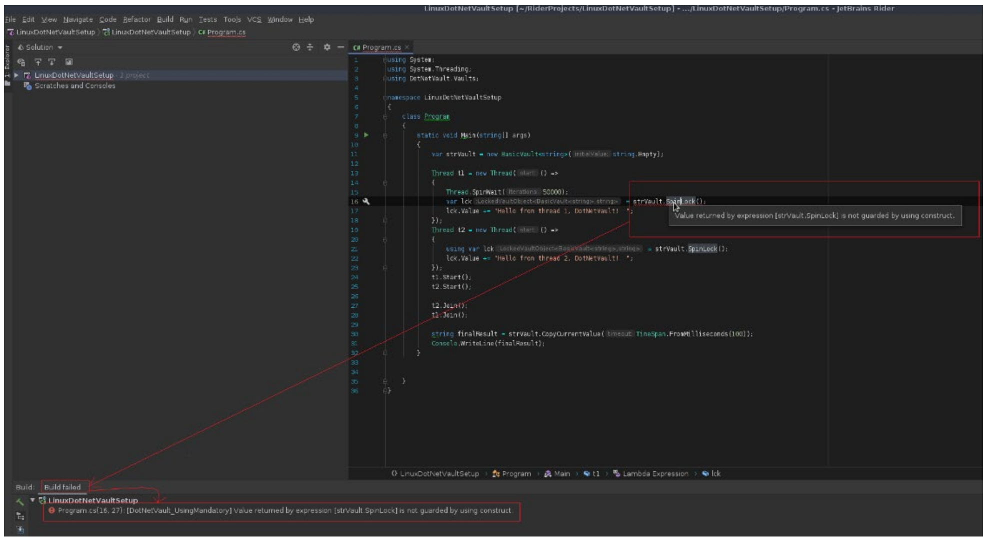
```
using System;
using System.Threading;
using DotNetVault.Vaults;

namespace LinuxDotNetVaultSetup
{
   class Program
   {
      static void Main(string[] args)
      {
         var strVault = new BasicVault<string>(string.Empty);

         Thread t1 = new Thread(() =>
         {
            Thread.SpinWait(50000);
            var lck = strVault.SpinLock();
            lck.Value += "Hello from thread 1, DotNetVault!  ";
         });
         Thread t2 = new Thread(() =>
         {
            using var lck = strVault.SpinLock();
            lck.Value += "Hello from thread 2, DotNetVault!  ";
         });

         t1.Start();
         t2.Start();
         t2.Join();
         t1.Join();

         string finalResult = 
strVault.CopyCurrentValue(TimeSpan.FromMilliseconds(100));
         Console.WriteLine(finalResult);
      }
   }
}
```

b. Click “Build” (or press Ctrl-Shift-B) as shown


LinuxDotNetVaultSetup [~/RiderP
File Edit View Navigate Code Refactor Build Run Tests Tools VCS Window Help
LinuxDotNetVaultSetup 〉 LinuxDotNet
Solution
Explorer
LinuxDotNetVaultSetup · 1 project
Scratches and Consoles
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Build Solution with Diagnostics
Rebuild Solution with Diagnostics
Change Solution Configuration...
Change Device...
C# Program.cs

```
1   using System;
2   using System.Threading;
3   using DotNetVault.Vaults;
4
5   namespace LinuxDotNetVaultSetup
6   {
7       class Program
```

c. If you have properly installed the DotNetVault package and Roslyn is enabled, the project **should <u>not</u> build.** If it builds, consult JetBrains documentation for how to enable Roslyn Analyzers. Do not attempt to use this library without static analysis enabled. Assuming it does not build, you should see the following as a result of your build attempt:


LinuxDotNetVaultSetup [~/RiderProjects/LinuxDotNetVaultSetup] - .../LinuxDotNetVaultSetup/Program.cs - JetBrains Rider
File Edit View Navigate Code Refactor Build Run Tests Tools VCS Window Help
LinuxDotNetVaultSetup 〉 LinuxDotNetVaultSetup 〉 Program.cs
Solution
LinuxDotNetVaultSetup · 1 project
Scratches and Consoles
C# Program.cs
LinuxDotNetVaultSetup › Program › Main › t1 › Lambda Expression › lck
Build: Build failed
LinuxDotNetVaultSetup

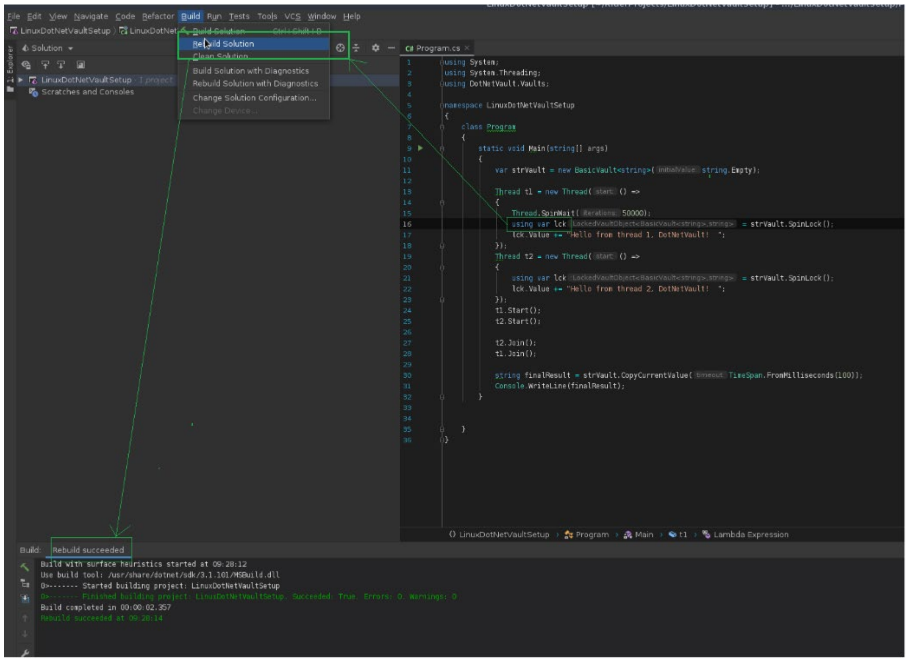
```
1       using System;
2       using System.Threading;
3       using DotNetVault.Vaults;
4
5       namespace LinuxDotNetVaultSetup
6       {
7           class Program
8           {
9               static void Main(string[] args)
10              {
11                  var strVault = new BasicVault<string>(string.Empty);
12
13                  Thread t1 = new Thread(() =>
14                  {
15                      Thread.SpinWait(50000);
16                      var lck = strVault.SpinLock();
17                      lck.Value += "Hello from thread 1, DotNetVault!  ";
18                  });
19                  Thread t2 = new Thread(() =>
20                  {
21                      using var lck = strVault.SpinLock();
22                      lck.Value += "Hello from thread 2, DotNetVault!  ";
23                  });
24                  t1.Start();
25                  t2.Start();
26
27                  t2.Join();
28                  t1.Join();
29
30                  string finalResult = strVault.CopyCurrentValue(TimeSpan.FromMilliseconds(100));
31                  Console.WriteLine(finalResult);
32              }
33
34
35          }
36      }
```

```
Value returned by expression [strVault.SpinLock] is not guarded by using construct.
```

```
Program.cs(16, 27): [DotNetVault_UsingMandatory] Value returned by expression [strVault.SpinLock] is not guarded by using construct.
```

d. The error is that you **must** guard the return value from a Lock() or SpinLock() method (or any other method whose return value you choose to annotate with the *UsingMandatory* attribute) with a using statement or declaration. Failure to ensure that the lock is promptly released would cause a serious error in your program … it would timeout whenever in the future you attempted to obtain the lock.

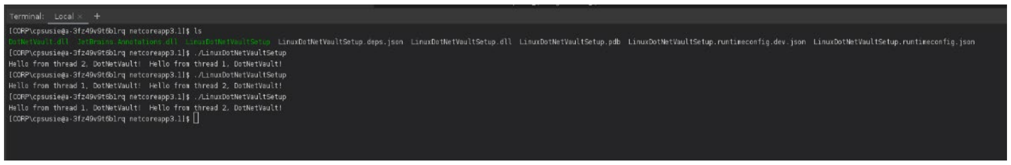
e. To fix the error, on line 16, change “var lck =…” to “using var lck =…” as shown then Build again. This time, the build should succeed as shown:


File Edit View Navigate Code Refactor Build Run Tests Tools VCS Window Help
Build Solution
Rebuild Solution
Clean Solution
Build Solution with Diagnostics
Rebuild Solution with Diagnostics
Change Solution Configuration...
Change Device...
Solution
LinuxDotNetVaultSetup
Scratches and Consoles
Program.cs
LinuxDotNetVaultSetup › Program › Main › t1 › Lambda Expression
Build: Rebuild succeeded

```
using System;
using System.Threading;
using DotNetVault.Vaults;

namespace LinuxDotNetVaultSetup
{
    class Program
    {
        static void Main(string[] args)
        {
            var strVault = new BasicVault<string>(string.Empty);

            Thread t1 = new Thread(() =>
            {
                Thread.SpinWait(50000);
                using var lck = strVault.SpinLock();
                lck.Value += "Hello from thread 1, DotNetVault!  ";
            });
            Thread t2 = new Thread(() =>
            {
                using var lck = strVault.SpinLock();
                lck.Value += "Hello from thread 2, DotNetVault!  ";
            });
            t1.Start();
            t2.Start();

            t2.Join();
            t1.Join();

            string finalResult = strVault.CopyCurrentValue(TimeSpan.FromMilliseconds(100));
            Console.WriteLine(finalResult);
        }


    }
}
```

```
Build with surface heuristics started at 09:28:12
Use build tool: /usr/share/dotnet/sdk/3.1.101/MSBuild.dll
0>------- Started building project: LinuxDotNetVaultSetup
0>------- Finished building project: LinuxDotNetVaultSetup. Succeeded: True. Errors: 0. Warnings: 0
Build completed in 00:00:02.357
Rebuild succeeded at 09:28:14
```

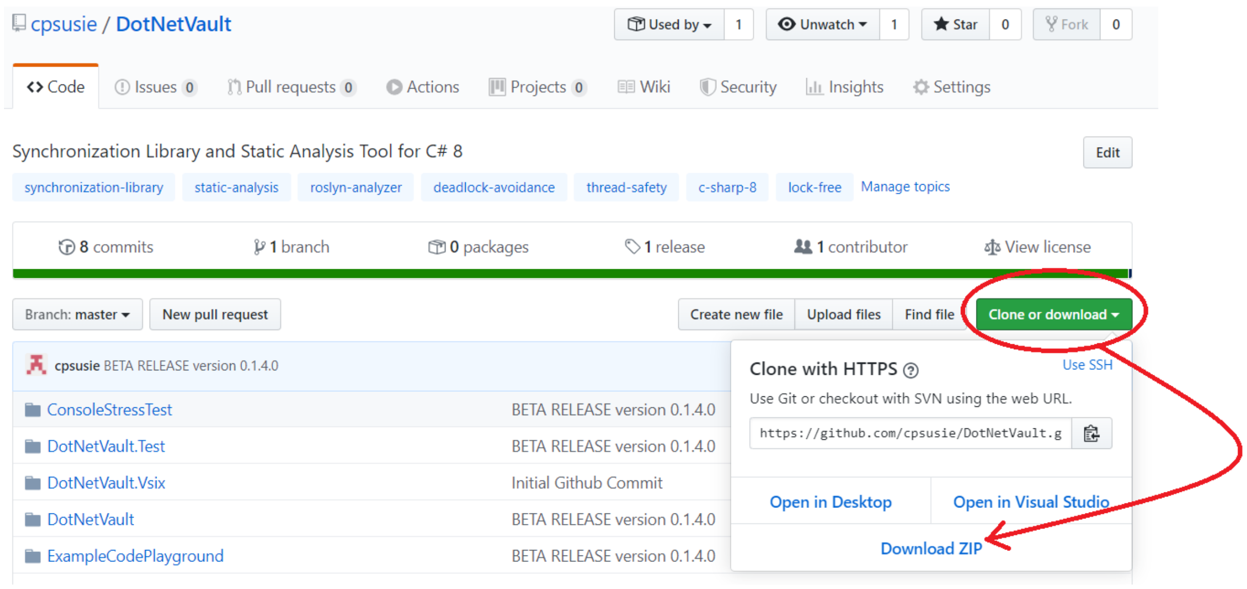
f. Now, you should be able to run the application as shown:



```
[CORP\cpsusie@a-3fz49v9t6blrq netcoreapp3.1]$ ls
DotNetVault.dll  JetBrains.Annotations.dll  LinuxDotNetVaultSetup  LinuxDotNetVaultSetup.deps.json  LinuxDotNetVaultSetup.dll  LinuxDotNetVaultSetup.pdb  LinuxDotNetVaultSetup.runtimeconfig.dev.json  LinuxDotNetVaultSetup.runtimeconfig.json
[CORP\cpsusie@a-3fz49v9t6blrq netcoreapp3.1]$ ./LinuxDotNetVaultSetup
Hello from thread 2, DotNetVault!  Hello from thread 1, DotNetVault!
[CORP\cpsusie@a-3fz49v9t6blrq netcoreapp3.1]$ ./LinuxDotNetVaultSetup
Hello from thread 1, DotNetVault!  Hello from thread 2, DotNetVault!
[CORP\cpsusie@a-3fz49v9t6blrq netcoreapp3.1]$ ./LinuxDotNetVaultSetup
Hello from thread 1, DotNetVault!  Hello from thread 2, DotNetVault!
[CORP\cpsusie@a-3fz49v9t6blrq netcoreapp3.1]$
```

8- Congratulations, you have successfully built an application using DotNetVault. Next, we will look at a QuickStart application that shows the very basics of what can be done with this library and its integrated static analyzer.

a. download the source code for DotNetVault from GitHub.



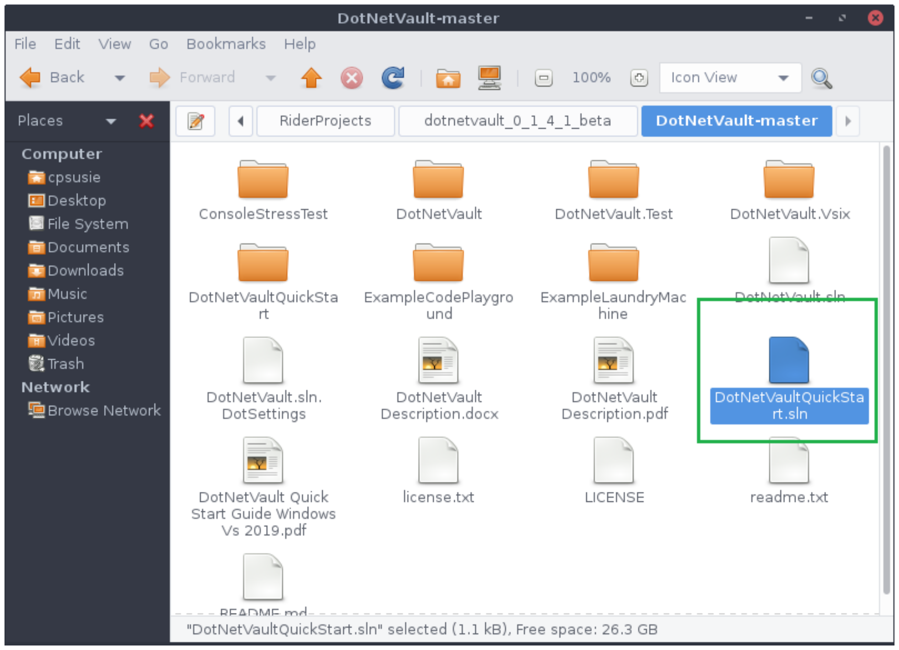
Synchronization Library and Static Analysis Tool for C# 8

cpsusie BETA RELEASE version 0.1.4.0

ConsoleStressTest	BETA RELEASE version 0.1.4.0
DotNetVault.Test	BETA RELEASE version 0.1.4.0
DotNetVault.Vsix	Initial Github Commit
DotNetVault	BETA RELEASE version 0.1.4.0
ExampleCodePlayground	BETA RELEASE version 0.1.4.0

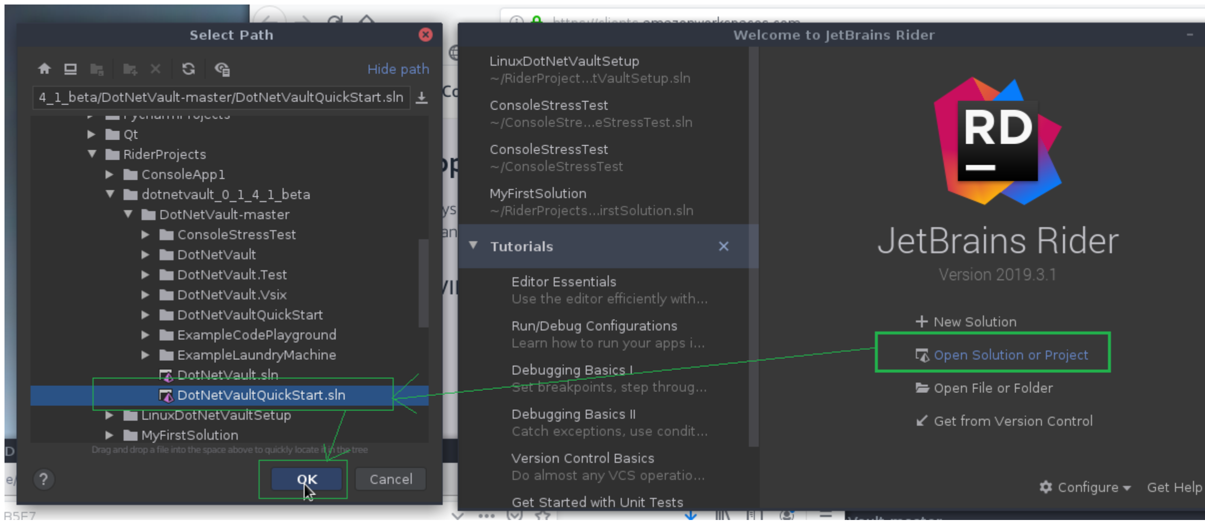
Use Git or checkout with SVN using the web URL.

b. Extract the zip file contents to a folder in a convenient place

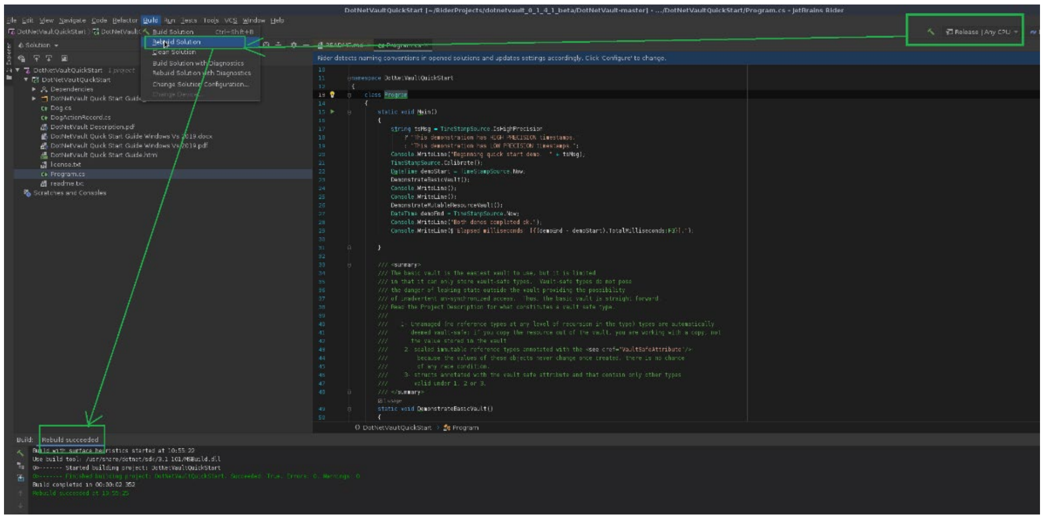


"DotNetVaultQuickStart.sln" selected (1.1 kB), Free space: 26.3 GB

c. Go into the folder structure and open the file "DotNetVaultQuickStart.sln" using Rider 2019.3.1+:



d. Build the solution.



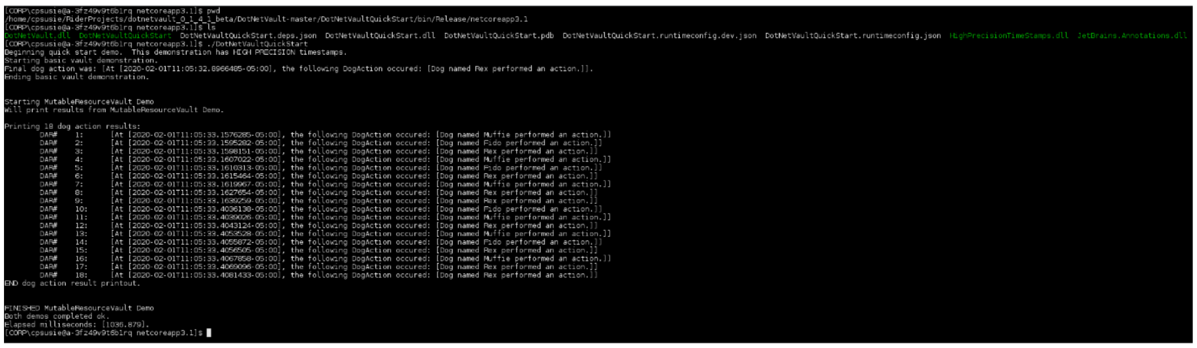
DotNetVaultQuickStart [~/RiderProjects/dotnetvault_0_1_4_1_beta/DotNetVault-master] - .../DotNetVaultQuickStart/Program.cs - JetBrains Rider
Build
Build Solution
Rebuild Solution
Clean Solution
Build Solution with Diagnostics
Rebuild Solution with Diagnostics
Change Solution Configuration...
Release | Any CPU
Rebuild succeeded
DotNetVaultQuickStart
Program

Rider detects naming conventions in opened solutions and updates settings accordingly. Click 'Configure' to change.

```
namespace DotNetVaultQuickStart
{
    class Program
    {
        static void Main()
        {
            string tsMsg = TimeStampSource.IsHighPrecision
                ? "This demonstration has HIGH PRECISION timestamps."
                : "This demonstration has LOW PRECISION timestamps.";
            Console.WriteLine("Beginning quick start demo.  " + tsMsg);
            TimeStampSource.Calibrate();
            DateTime demoStart = TimeStampSource.Now;
            DemonstrateBasicVault();
            Console.WriteLine();
            Console.WriteLine();
            DemonstrateMutableResourceVault();
            DateTime demoEnd = TimeStampSource.Now;
            Console.WriteLine("Both demos completed ok.");
            Console.WriteLine($"Elapsed milliseconds: [{(demoEnd - demoStart).TotalMilliseconds:F3}].");

        }

        /// <summary>
        /// The basic vault is the easiest vault to use, but it is limited
        /// in that it can only store vault-safe types.  Vault-safe types do not pose
        /// the danger of leaking state outside the vault providing the possibility
        /// of inadvertent un-synchronized access.  Thus, the basic vault is straight forward
        /// Read the Project Description for what constitutes a vault safe type.
        ///
        ///     1- Unmanaged (no reference types at any level of recursion in the type) types are automatically
        ///        deemed vault-safe: if you copy the resource out of the vault, you are working with a copy, not
        ///        the value stored in the vault
        ///     2- sealed immutable reference types annotated with the <see cref="VaultSafeAttribute"/>
        ///         because the values of these objects never change once created, there is no chance
        ///         of any race condition.
        ///     3- structs annotated with the vault safe attribute and that contain only other types
        ///         valid under 1, 2 or 3.
        /// </summary>
        static void DemonstrateBasicVault()
        {
```

```
Build with surface heuristics started at 10:55:22
Use build tool: /usr/share/dotnet/sdk/3.1.101/MSBuild.dll
0>------- Started building project: DotNetVaultQuickStart
0>------- Finished building project: DotNetVaultQuickStart. Succeeded: True. Errors: 0. Warnings: 0
Build completed in 00:00:02.352
Rebuild succeeded at 10:55:25
```

e. Run the solution and check output, which should be something like this:



```
[CORP\cpsusie@a-3fz49v9t6b1rq netcoreapp3.1]$ pwd
/home/cpsusie/RiderProjects/dotnetvault_0_1_4_1_beta/DotNetVault-master/DotNetVaultQuickStart/bin/Release/netcoreapp3.1
[CORP\cpsusie@a-3fz49v9t6b1rq netcoreapp3.1]$ ls
DotNetVault.dll  DotNetVaultQuickStart  DotNetVaultQuickStart.deps.json  DotNetVaultQuickStart.dll  DotNetVaultQuickStart.pdb  DotNetVaultQuickStart.runtimeconfig.dev.json  DotNetVaultQuickStart.runtimeconfig.json  HighPrecisionTimeStamps.dll  JetBrains.Annotations.dll
[CORP\cpsusie@a-3fz49v9t6b1rq netcoreapp3.1]$ ./DotNetVaultQuickStart
Beginning quick start demo.  This demonstration has HIGH PRECISION timestamps.
Starting basic vault demonstration.
Final dog action was: [At [2020-02-01T11:05:32.8966485-05:00], the following DogAction occured: [Dog named Rex performed an action.]].
Ending basic vault demonstration.


Starting MutableResourceVault Demo
Will print results from MutableResourceVault Demo.

Printing 18 dog action results:
        DAR#    1:      [At [2020-02-01T11:05:33.1576285-05:00], the following DogAction occured: [Dog named Muffie performed an action.]]
        DAR#    2:      [At [2020-02-01T11:05:33.1595282-05:00], the following DogAction occured: [Dog named Fido performed an action.]]
        DAR#    3:      [At [2020-02-01T11:05:33.1598151-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
        DAR#    4:      [At [2020-02-01T11:05:33.1607022-05:00], the following DogAction occured: [Dog named Muffie performed an action.]]
        DAR#    5:      [At [2020-02-01T11:05:33.1610313-05:00], the following DogAction occured: [Dog named Fido performed an action.]]
        DAR#    6:      [At [2020-02-01T11:05:33.1615464-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
        DAR#    7:      [At [2020-02-01T11:05:33.1619967-05:00], the following DogAction occured: [Dog named Muffie performed an action.]]
        DAR#    8:      [At [2020-02-01T11:05:33.1627654-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
        DAR#    9:      [At [2020-02-01T11:05:33.1639259-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
        DAR#    10:     [At [2020-02-01T11:05:33.4036138-05:00], the following DogAction occured: [Dog named Fido performed an action.]]
        DAR#    11:     [At [2020-02-01T11:05:33.4039026-05:00], the following DogAction occured: [Dog named Muffie performed an action.]]
        DAR#    12:     [At [2020-02-01T11:05:33.4043124-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
        DAR#    13:     [At [2020-02-01T11:05:33.4053528-05:00], the following DogAction occured: [Dog named Muffie performed an action.]]
        DAR#    14:     [At [2020-02-01T11:05:33.4055872-05:00], the following DogAction occured: [Dog named Fido performed an action.]]
        DAR#    15:     [At [2020-02-01T11:05:33.4056505-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
        DAR#    16:     [At [2020-02-01T11:05:33.4067858-05:00], the following DogAction occured: [Dog named Muffie performed an action.]]
        DAR#    17:     [At [2020-02-01T11:05:33.4069096-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
        DAR#    18:     [At [2020-02-01T11:05:33.4081433-05:00], the following DogAction occured: [Dog named Rex performed an action.]]
END dog action result printout.


FINISHED MutableResourceVault Demo
Both demos completed ok.
Elapsed milliseconds: [1036.879].
[CORP\cpsusie@a-3fz49v9t6b1rq netcoreapp3.1]$ █
```

f. As you can see, from Main(), the demo demonstrates the use of the BasicVault, which stores VaultSafe objects and the MutableResourceVault which stores objects that are not VaultSafe.

   i. VaultSafe objects are ones that do not require effort to keep isolated. They include

      1. Unmanaged value types (e.g. long, enums, DateTime, TimeSpan, etc.)
      2. Sealed immutable reference types that are annotated with the VaultSafe attribute (string is automatically considered vault-safe; Immutable collections from System.Collections.Immutable that have only vault-safe type arguments are also considered vault-safe automatically).
      3. Value types that are annotated with the VaultSafe attribute and contain only other types that comply with #1, 2 and 3

4. Unmanaged value types, strings and qualifying immutable collections are considered VaultSafe without need for annotation with the VaultSafe attribute.

These objects are easy to isolate because copies of them are either true deep copies (unmanaged types), totally immutable reference types (no danger of a stored reference changing the protected value), or value types that contain only types that are unmanaged value types and immutable reference types. The resource protected in the `DemonstrateBasicVault()` in *Program.cs* is a *DogActionRecord*, found in DogActionRecord.cs.

DogActionRecord is VaultSafe because:

1. It is annotated with the VaultSafe attribute
2. It is a value type with field members that include
    a. An unmanaged value type (DateTime) and
    b. A sealed immutable reference type (string)

g. The `DemonstrateBasicVault()` in *Program.cs* creates a BasicVault<DogActionRecord>, a vault that protects a DogActionRecord and can be used to obtain locks for synchronized access to the value. It then creates a List of Dogs (Dog.cs) then calls their `DoDogActions()` methods, which causes each to spawn a thread that obtains a lock on the vault and overwrites the value stored therein with an action.

The syntax is straight forward:

```
while (numActions-- > 0)
{
    {
        //using the basicvault's locked resource object is easy: get or set.
        //we don't have to worry about copying the value out of the vault -- because it is vault-safe,
        //the value is effectively a deep copy (and the string member is immutable and sealed).
        //The copy cannot be used to affect the value stored in the vault after the lock is released.
        using var lck = _recordVault.SpinLock();
        lck.Value = new DogActionRecord($"Dog named {Name} performed an action.");
    }
    Thread.Sleep(TimeSpan.FromMilliseconds(1));
}
```

Figure 1

At the end of the demonstration, the main thread obtains the lock and prints out the *DogActionRecord* that happens to be there (because the order is non-deterministic, there will be different results):

```
//SpinLock is a busy wait; Lock sleeps.
//You will not deadlock -- If you cannot obtain the resource within the specified time period,
//an exception of type TimeoutException may be thrown.  The parameterless Lock and SpinLock
//methods use a default timeout period.  You may also specify a positive timespan as your own
//timeout period.  Alternatively, you may supply a cancellationtoken, either alone or in conjunction
//with the timeout period.  If just the cancellation token is supplied, attempts to obtain the lock
//will continue until it is obtained or cancellation request is propagated to token.  If token and timespan
//are supplied attempts will continue until the earlier of:
// 1- resource is obtained
// 2- cancel request propagated to token
// 3- timeout period expires
using var lck = actionVault.SpinLock();
Console.WriteLine($"Final dog action was: [{lck.Value.ToString()}].");
```

Figure 2

h. The `DemonstrateMutableResourceVault()` method in Program.cs shows protection of a resource that IS NOT vault-safe. It is a SortedSet of DogActionRecords. The dogs in this demo, instead of overwriting a single DogActionRecord, will add their DogActionRecords to the SortedSet (which maintains them ordered by timestamp, convenient for printing in order at end).

   Resources that are not VaultSafe are protected by a MutableResourceVault, the lock objects of which are more restrictive because they need to make sure that only VaultSafe types are passed into the protected resource or received out from the protected resource; otherwise, mutable state (to which a reference may exist outside) could mingle with the protected resource or mutable state inside the protected resource could leak to the outside and cause unsynchronized access to the protected resource.

The syntax for creating a MutableResourceVault is shown:

```
//You supply the ctor for the protected resource to the mutable resource vault factory,
//that way the resource vault will create it and there is no way to get to it outside the vault
//do not use a lambda that simply provides access to existing object: there is no reasonable
//way for the analyzer to protect you.  Pass the ctor in as a lambda, the vault will create
//and guard the resource.  The provision of the timespan argument is optional.
//If you do not provide one, a default value will be used.
//See comments in the BasicVault demonstration regarding disposing VAULTS
using (var mutableResourceVault =
    MutableResourceVault<SortedSet<DogActionRecord>>.CreateMutableResourceVault(
        () => new SortedSet<DogActionRecord>(), TimeSpan.FromSeconds(1)))
```

Figure 3

Accessing the mutable resource through the lock is mediated by delegates:

```
while (numActions-- > 0)
{
    {
        //using the mutable resource vault is a little more tricky.  All inputs to and outputs from the non-vault-safe
        //resource must THEMSELVES be vault-safe even though the protected resource is not.  Thus, access
        //to the protected mutable resource is mediated by delegates with special attributes that are meaningful
        //to the integrated static analyzer.  Using non-vault safe parameters or capturing or referencing non-vault
        //safe values (other than the protected resource itself) is detected by the analyzer, which will refuse
        //to compile the code until you use a vault-safe alternative.  If use of shared mutable state were allowed
        //the analyzer would have no reasonable way to prevent shared mutable state from escaping the vault or
        //shared mutable state accessible from outside the vault from changing values inside of it.
        using var lck = _vault.SpinLock();
        bool addedOk= lck.ExecuteMixedOperation(
            (ref SortedSet<DogActionRecord> res, in DogActionRecord record) => res.Add(record),
            new DogActionRecord($"Dog named {Name} performed an action."));
        //See the Project Description Pdf for a full description of the delegates used to update protected mutable resources
        //and for how to use extension methods and/or custom vault objects to make this less cumbersome if used very
        //frequently
        if (!addedOk)
        {
            Console.Error.WriteLineAsync(
                $"At {TimeStampSource.Now:O}, a dog action record could not be added to the sorted set.");
        }
    }
    //Help keep the output randomized not dominated by same dog
    Thread.SpinWait(_rgen.Value.Next(1, 2500));
    Thread.Sleep(TimeSpan.FromMilliseconds(1));
}
```

Figure 4

At the end of the simulation, a lock is acquired that returns the SortedSet to the main thread as an ImmutableSortedSet<DogActionRecord> as shown:

```
    //The resource itself is mutable -- a SortedSet, though the items it contains are VaultSafe.
    //It is not recommended to store COLLECTIONS of items that are not vault safe because there is almost
    //no way to ensure their isolation (and thus freedom from race conditions).  If the resource itself, however,
    //such a string, a DateTime, a DogActionRecord, a long, an enum, etc is VaultSafe itself, it is easy to store
    //them in a mutable collection protected by the a MutableResourceVault.
    ImmutableSortedSet<DogActionRecord> results;
    {
        using var lck = mutableResourceVault.SpinLock();
        results = lck.ExecuteQuery((in SortedSet<DogActionRecord> res) => res.ToImmutableSortedSet());
    } //lock is released here

    finalResults = ProcessResults(results);
}
Console.WriteLine("Will print results from MutableResourceVault Demo.");
Console.WriteLine(Environment.NewLine + finalResults + Environment.NewLine);
Console.WriteLine("FINISHED MutableResourceVault Demo");
```

Figure 5

i. Further resources.
    i. The DotNetVault itself comes with an example of a vault customized to protect StringBuilder resources without resort to the inconvenient delegate syntax. It contains directions for making your own customized vaults and locked resource objects.
    ii. Included in the source repository is the ConsoleStressTest using .NET Core 3.1. It can be run in a Linux or Windows environment and demonstrates the effectiveness of DotNetVault at protecting a mutable resource with a high degree of thread contention.
    iii. The LaundryMachine.sln and the LaundryStressTest project therein requires Windows because it uses WPF. It demonstrates the usage of many vaults and many threads in contention in a highly (unnecessarily so, but good to demonstrate DotNetVault's usefulness even in convoluted scenarios) complex multithreaded state machine scenario where LaundryMachines have their own threads and loader and unloader robots (also with their own threads) contend for access to laundry machine. The simulation runs until all soiled articles are cleaned. The loader robots put dirty laundry into one of the machines then start the cycle, the unloader robots take dirty laundry from the machine and put them in the clean bin. The robots constantly contend with each other for access to the machine and since each Laundry Machines state machine thread is independent, the robots also contend with the state machine threads as well as each other to access the machines.
    iv. The ExampleCodePlayground project is used to get a feel for the static analyzer's rules and how they work.
    v. There is a unit test project primarily oriented around ensuring that the static analyzer rules work properly.

j. The DotNetVault Description.pdf provides detailed information on the DotNetVault, the vaults and LockedResourceObjects, its reasoning and static analysis rules as well as examples.