# Task manager (C#)

Technical design

## 1 Introduction

Goal: A task manager (for the software development area) should be implemented using C# programming language.

## 2 Requirements and Implementation

The solution consists of 3 projects:

1. TaskManagerCore – implements business logic
2. TaskManagerConsole – implements Console UI
3. TaskManagerWinForms – implements GUI

There is also the UnitTests project – implements Unit Tests

### 2.1 OOP

#### 2.1.1 At least 10 classes

See the class diagram in Appendix A.

#### 2.1.2 Encapsulation

Almost each class contain at least one private Field or Method. Private field names start with “v”, according to SC C# Style Guide. Usually such a fields are used in a pair with a public Property, e.g.:

```
        private string vUserName = "";
        public string UserName { get { return vUserName; } }
```

#### 2.1.3 Polymorphism

There are few classes which override ancestor’s methods (excluding the “standard” IDisposable implementation which overrides Dispose()):

1. AuthForms.AuthenticateUser() overrides Authenticator.AuthenticateUser() – implements user authentication via WinForms GUI
2. AuthConsole.AuthenticateUser() overrides Authenticator.AuthenticateUser() – implements user authentication via Console

#### 2.1.4 Inheritance

The solution contains 3 inheritance trees:

1. User→{Admin, Manager, Developer}
2. BaseTask→{DevTask, TestTask}
3. Authenticator→{AuthConsole, AuthForms}

#### 2.1.5 Interfaces

The solution contains 2 interfaces:

1. IStorage – general interface to the Data Access Layer
2. IAuthenticator – interface to the user authentication subsystem

There is also a placeholder for the ITask interface.

Interface IDisposable has been implemented for the UserSessions class.

### 2.1.6 Abstract class

The solution contains 2 abstract classes: User and BaseTask

## 2.2 Multithreading and Events/Delegates

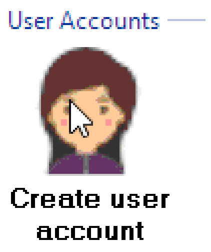
1. Class UserSession implements the DoHandleActions() method which runs in a separate thread:

```
vActionHandler = new Task(DoHandleActions, (object)this,
SessionCancellationTokenSource.Token,
TaskCreationOptions.AttachedToParent |
TaskCreationOptions.PreferFairness);
vActionHandler.Start();
```

   The UserSessions could be of the following types (UserSessionTypes):
   1. Automatic – background session, no live user interface
   2. InertactiveConcole – session, which interacts with a live user via the Console
   3. InteractiveGUI – session, which interacts with a live user via WinForms GUI

   DoHandleActions() method runs a loop checking the CancellationToken. Inside the loop it checks the UserTask queue:

```
private ConcurrentQueue<UserTask> vUserTasks = new
ConcurrentQueue<UserTask>();
```

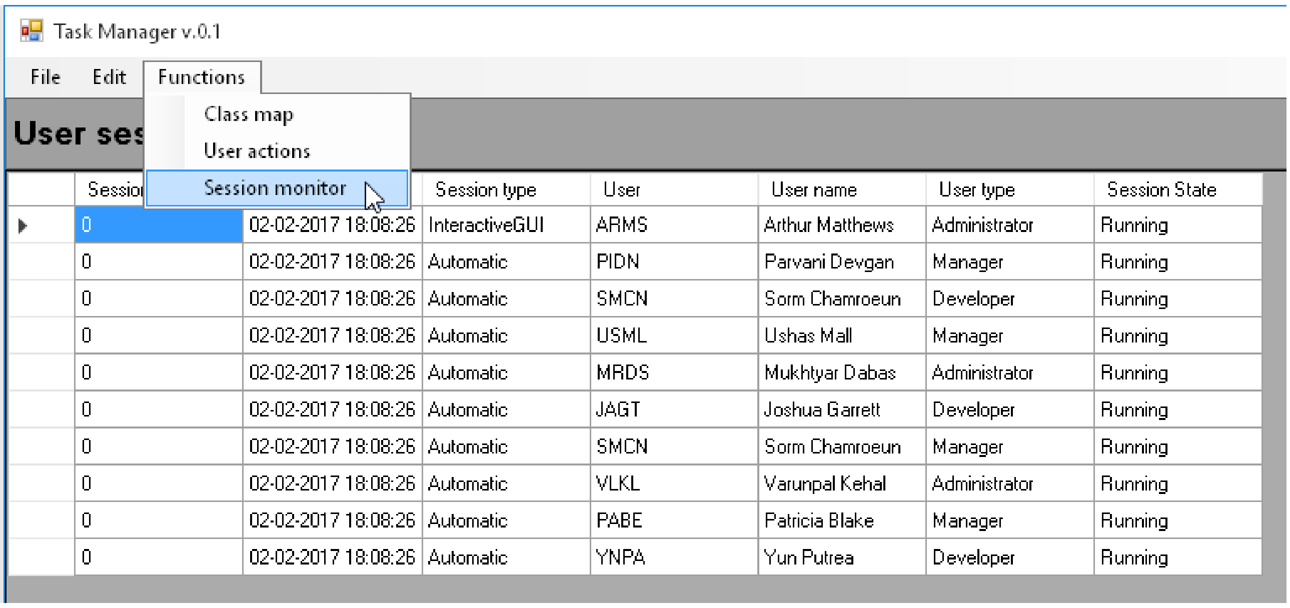


   In the GUI implementation a UserTask is enqueued into the current user's queue when the user presses on an action item. Currently a UserTask contains a MethodInfo variable (all the GUI actions are bound to some class and method, see the Custom Attributes section).

   If the queue contains a task, it takes it and tries to request the task parameters for the MethodInfo method via DoAskParameters **delegate**.

   In the GUI implementation the DoAskParameters call path looks like that: UserSession.DoAskParameters→TaskManager.DoAskParameters→Form1.DoAskParameters---<**this.Invoke()**>--->Form1.FormAskParameters, which makes it possible to access the GUI elements from another thread. There is a DoAskParametersAutomatic

   DoHandleActions() method also raises the `SessionChangedEvent` **event**, once in the beginning, another in the end. It has the same call path and is used to update the Status Bar with the information about running UserSessions.
   It also used to switch the GUI either to the Authorisation page, or to the Shutdown/Logoff page.
2. When a user logs on for the first time, 10 background user sessions are started via UserSession.Run method which creates a separate Task. In the Console version one can see the test output from these threads.



| | Sessio | | Session type | User | User name | User type | Session State |
|---|---|---|---|---|---|---|---|
| ▶ | 0 | 02-02-2017 18:08:26 | InteractiveGUI | ARMS | Arthur Matthews | Administrator | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | PIDN | Parvani Devgan | Manager | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | SMCN | Sorn Chamroeun | Developer | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | USML | Ushas Mall | Manager | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | MRDS | Mukhtyar Dabas | Administrator | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | JAGT | Joshua Garrett | Developer | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | SMCN | Sorn Chamroeun | Manager | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | VLKL | Varunpal Kehal | Administrator | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | PABE | Patricia Blake | Manager | Running |
| | 0 | 02-02-2017 18:08:26 | Automatic | YNPA | Yun Putrea | Developer | Running |

3. On Exit each UserSession's CancellationToken is raised, so each user thread stops after completion of current task.
4. Async/await is used in the GUI for long-running tasks, e.g.:

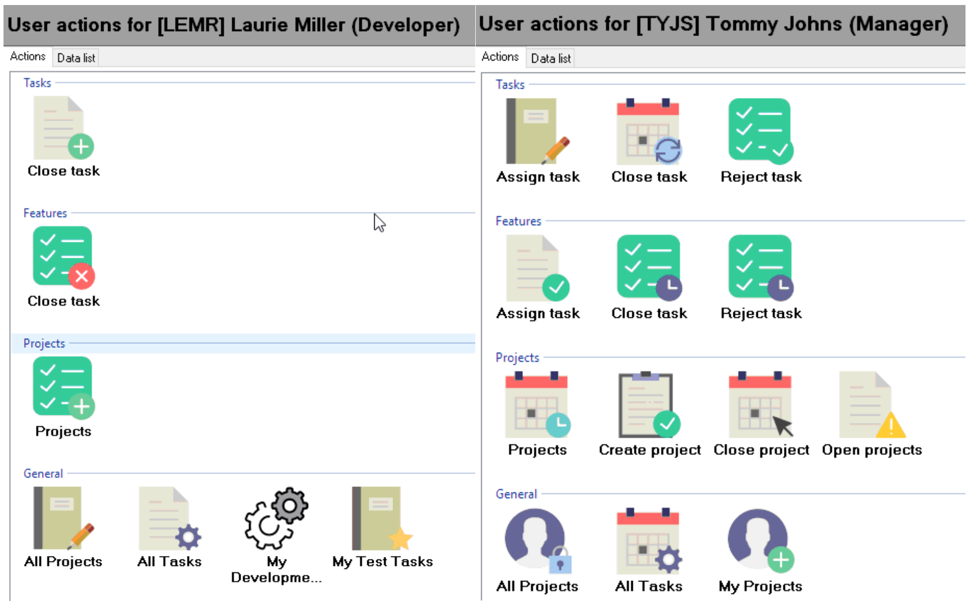
```
Task taskShutdown = Task.Run(() => vTaskManager.Shutdown());
await taskShutdown;
```

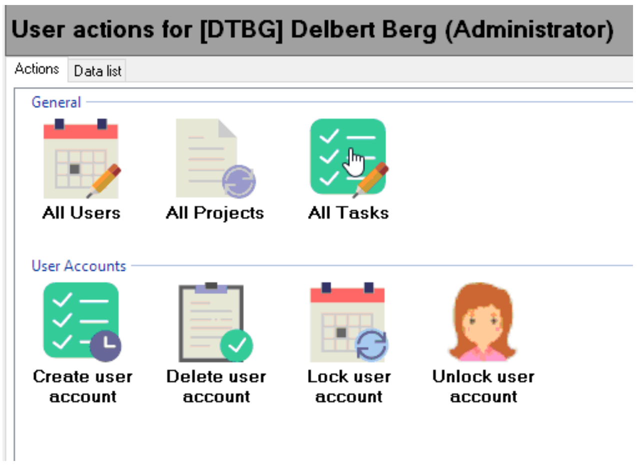
## 2.3 Custom Attributes

The UserActions attribute is implemented:

```
UserAction(string description, UserTypes[] usersAllowed)
```

It is applied to classes and methods. All other cases are ignored. If the attribute is applied to a class, it defines an Action Group for the GUI. If being applied to a method, it defines an Action. Also usersAllowed fields defines the UserTypes (Administrator, Developer,Manager) which have an access to the Group/Action (images are assigned randomly):





Typical usage looks like this:

```
[UserAction("Tasks", new UserTypes[] {
UserTypes.Administrator,UserTypes.Manager,UserTypes.Developer})]
    public abstract class BaseTask {
        [UserAction("Assign task", new UserTypes[] { UserTypes.Manager})]
        public void Assign() { }

        [UserAction("Close task", new UserTypes[] { UserTypes.Manager,
UserTypes.Developer })]
        public void Close() { }

        [UserAction("Reject task", new UserTypes[] { UserTypes.Manager })]
        public void Reject() { }
    }
```

## 2.4 Custom Exceptions

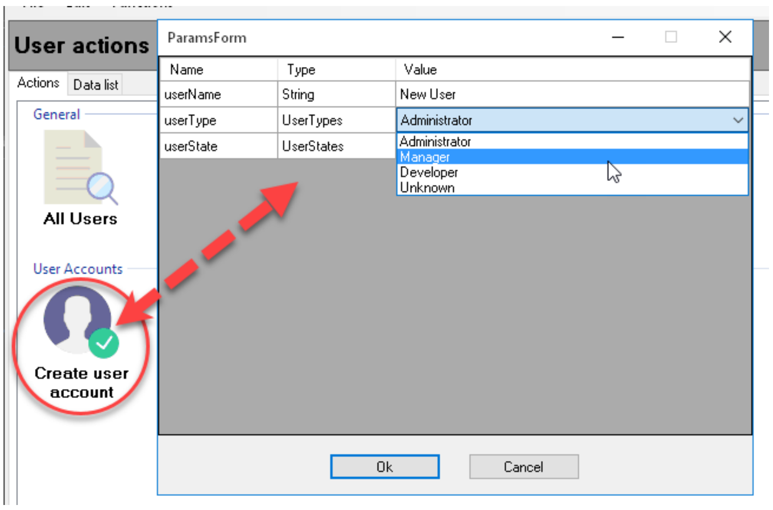
There is only one custom exception in the solution:

```
public class UserException:Exception
```

## 2.5 Reflection (class map)

Reflection is used widely in the solution.

1. There is the ReflectionInfo class which provides different application metadata.
2. The GUI Groups and Actions are based on the UserAction custom attribute.
3. When a user clicks on an Action, it passes the MethodInfo instance down to the working thread, where it’s handled further.
4. The ParamsForm form accepts the MethodInfo instance, parses it for a method parameters, and shows the parameter grid:



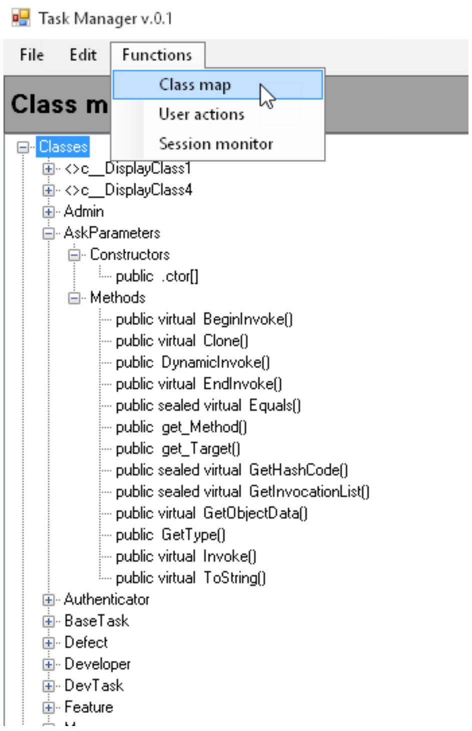
| Name | Type | Value |
|---|---|---|
| userName | String | New User |
| userType | UserTypes | Administrator |
| userState | UserStates | |

Administrator
Manager
Developer
Unknown

The corresponding code is:

```
[UserAction("Create user account", new UserTypes[] { UserTypes.Administrator })]
public User Create(string userName, UserTypes userType, UserStates userState) {
```

5. There is a dedicated “Class map” page in the application:



```
Classes
  <>c__DisplayClass1
  <>c__DisplayClass4
  Admin
  AskParameters
    Constructors
      public .ctor()
    Methods
      public virtual BeginInvoke()
      public virtual Clone()
      public DynamicInvoke()
      public virtual EndInvoke()
      public sealed virtual Equals()
      public get_Method()
      public get_Target()
      public sealed virtual GetHashCode()
      public sealed virtual GetInvocationList()
      public virtual GetObjectData()
      public GetType()
      public virtual Invoke()
      public virtual ToString()
  Authenticator
  BaseTask
  Defect
  Developer
  DevTask
  Feature
```

## 2.6 LINQ

There are few places in the solution where LINQ is used. Typically it’s used for reordering and some filtering:

1. 
```
int sessionCount = (from UserSession us in
vTaskManager.UserSessions where (us.ActionHandler.Status ==
TaskStatus.Running) select us).ToList().Count;
```

```
2. var constInfo = new List<MethodBase>(from rtCons in
   refType.GetConstructors() orderby rtCons.Name select rtCons);
   var metInfo = new List<MethodBase>((from mt in refType.GetMethods()
   orderby mt.Name select mt));
   foreach (var mi in constInfo.Union(metInfo)) {…
```

## 2.7 Patterns

The Factory pattern is implemented in the Users class:

```
public static User Factory(string userName, UserTypes userType) {
    User user = null;
    switch (userType) {
        case UserTypes.Administrator: { user = new Admin(userName); }
        break;
        case UserTypes.Developer: { user = new Developer(userName); }
        break;
        case UserTypes.Manager: { user = new Manager(userName); }
        break;
        default:
        break;
    }
    return user;
}
```

## 2.8 Unit Testing

There is a small Unit Test project in the solution:

```
namespace UnitTests {
    [TestClass]
    public class UnitTest1 {
        [TestMethod]
        public void TestMethod1() {
            Storage storage = new Storage();
            Assert.AreEqual(0, storage.ListUsers().Count);
            storage.Load();
            Assert.AreNotEqual(0, storage.ListUsers().Count);
            //Assert.Fail("Oops");
        }
    }
}
```

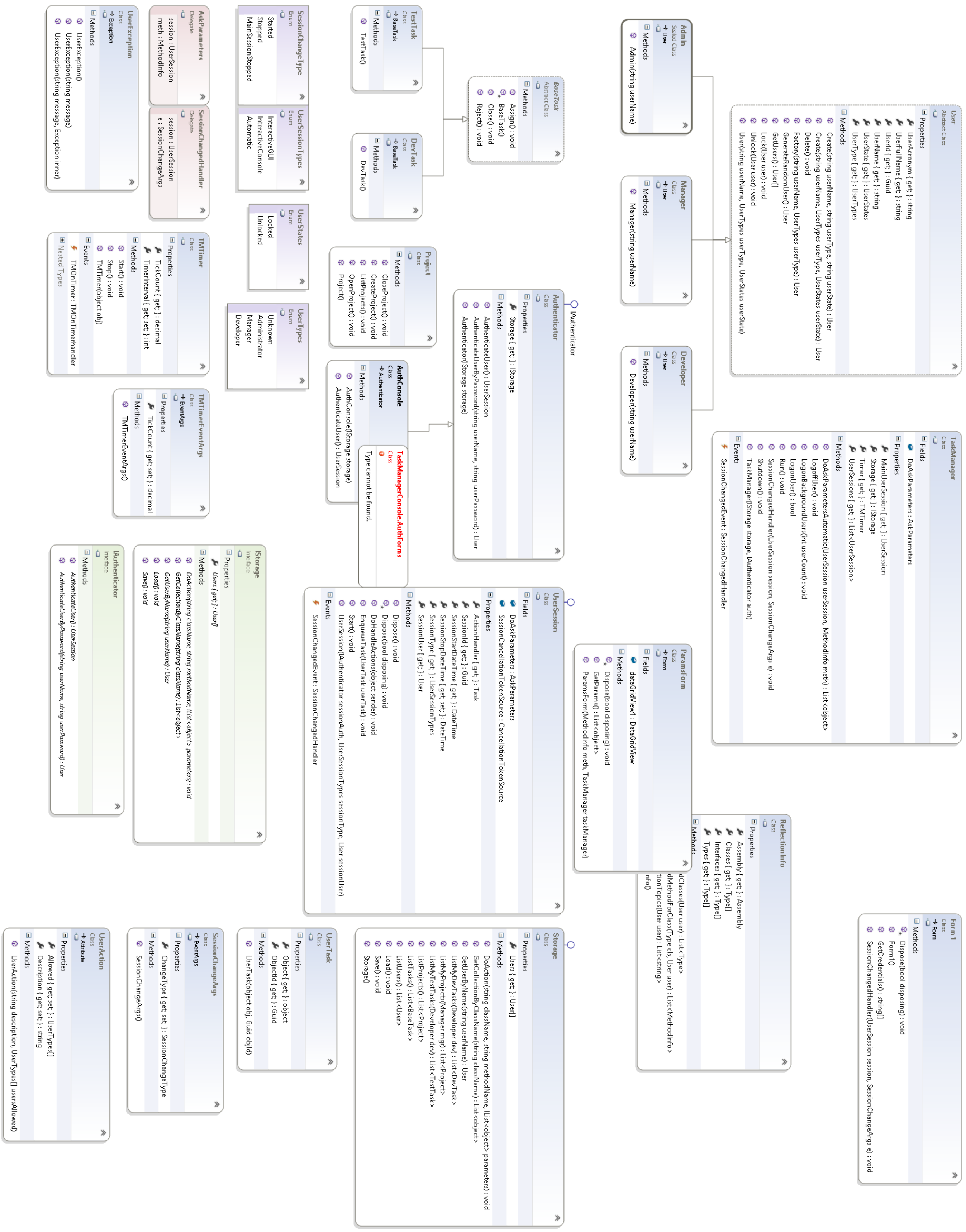
## 2.9 MVC

(View: Form1) ➔ (Controller: TaskManager, UserSession) ➔(Model: Storage)

## 2.10 GitHub

https://www.github.com/atom80/CSharp-course

# 3 Appendix A


UserException
Class
Exception
Methods
UserException()
UserException(string message)
UserException(string message, Exception inner)
AskParameters
Delegate
session : UserSession
meth : MethodInfo
SessionChangedHandler
Delegate
session : UserSession
e : SessionChangeArgs
SessionChangeType
Enum
Started
Stopped
MainSessionStopped
UserSessionTypes
Enum
InteractiveGUI
InteractiveConsole
Automatic
UserStates
Enum
Locked
Unlocked
UserTypes
Enum
Unknown
Administrator
Manager
Developer
TestTask
Class
BaseTask
Methods
TestTask()
DevTask
Class
BaseTask
Methods
DevTask()
BaseTask
Abstract Class
Methods
Assign() : void
BaseTask()
Close() : void
Reject() : void
Project
Class
Methods
CloseProject() : void
CreateProject() : void
ListProjects() : void
OpenProject() : void
Project()
TMTimer
Class
Properties
TickCount { get; set; } : decimal
TimeInterval { get; set; } : int
Methods
Start() : void
Stop() : void
TMTimer(object obj)
Events
TMOnTimer : TMOnTimerHandler
Nested Types
TMTimerEventArgs
Class
EventArgs
Properties
TickCount { get; set; } : decimal
Methods
TMTimerEventArgs()
Authenticator
Class
IAuthenticator
Properties
Storage { get; } : IStorage
Methods
AuthenticateUser() : UserSession
AuthenticateUserByPassword(string userName, string userPassword) : User
Authenticator(IStorage storage)
AuthConsole
Class
Authenticator
Methods
AuthConsole(IStorage storage)
AuthenticateUser() : UserSession
TaskManagerConsole.AuthForms
Class
Type cannot be found.
Admin
Sealed Class
User
Methods
Admin(string userName)
Manager
Class
User
Methods
Manager(string userName)
Developer
Class
User
Methods
Developer(string userName)
User
Abstract Class
Properties
UserAcronym { get; } : string
UserFullName { get; } : string
UserId { get; } : Guid
UserName { get; } : string
UserState { get; } : UserStates
UserType { get; } : UserTypes
Methods
Create(string userName, string userType, string userState) : User
Create(string userName, UserTypes userType, UserStates userState) : User
Delete() : void
Factory(string userName, UserTypes userType) : User
GenerateRandomUser() : User
GetUsers() : User[]
Lock(User user) : void
Unlock(User user) : void
User(string userName, UserTypes userType, UserStates userState)
TaskManager
Class
Fields
DoAskParameters : AskParameters
Properties
MainUserSession { get; } : UserSession
Storage { get; } : IStorage
Timer { get; } : TMTimer
UserSessions { get; } : List&lt;UserSession&gt;
Methods
DoAskParametersAutomatic(UserSession userSession, MethodInfo meth) : List&lt;object&gt;
LogoffUser() : void
LogonBackgroundUsers(int userCount) : void
LogonUser() : bool
Run() : void
SessionChangedHandler(UserSession session, SessionChangeArgs e) : void
Shutdown() : void
TaskManager(IStorage storage, IAuthenticator auth)
Events
SessionChangedEvent : SessionChangedHandler
IStorage
Interface
Properties
Users { get; } : User[]
Methods
DoAction(string className, string methodName, IList&lt;object&gt; parameters) : void
GetCollectionByClassName(string className) : List&lt;object&gt;
GetUserByName(string userName) : User
Load() : void
Save() : void
IAuthenticator
Interface
Methods
AuthenticateUser() : UserSession
AuthenticateUserByPassword(string userName, string userPassword) : User
UserSession
Class
Fields
DoAskParameters : AskParameters
SessionCancellationTokenSource : CancellationTokenSource
Properties
ActionHandler { get; } : Task
SessionId { get; } : Guid
SessionStartDateTime { get; set; } : DateTime
SessionStopDateTime { get; set; } : DateTime
SessionType { get; } : UserSessionTypes
SessionUser { get; } : User
Methods
Dispose() : void
Dispose(bool disposing) : void
DoHandleAction(object sender) : void
EnqueueTask(UserTask userTask) : void
Start() : void
UserSession(IAuthenticator sessionAuth, UserSessionTypes sessionType, User sessionUser)
Events
SessionChangedEvent : SessionChangedHandler
ParamForm
Class
Form
Fields
dataGridView1 : DataGridView
Methods
Dispose(bool disposing) : void
GetParams() : List&lt;object&gt;
ParamForm(MethodInfo meth, TaskManager taskManager)
ReflectionInfo
Class
Properties
Assembly { get; } : Assembly
Classes { get; } : Type[]
Interfaces { get; } : Type[]
Types { get; } : Type[]
Methods
dClasses(User user) : List&lt;Type&gt;
dMethodForClass(Type cls, User user) : List&lt;MethodInfo&gt;
tionTopics(User user) : List&lt;string&gt;
nfo()
Form1
Class
Form
Methods
Dispose(bool disposing) : void
Form1()
GetCredentials() : string[]
SessionChangedHandler(UserSession session, SessionChangeArgs e) : void
UserAction
Class
Attribute
Properties
Allowed { get; set; } : UserType[]
Description { get; set; } : string
Methods
UserAction(string description, UserTypes[] usersAllowed)
SessionChangeArgs
Class
EventArgs
Properties
ChangeType { get; set; } : SessionChangeType
Methods
SessionChangeArgs()
UserTask
Class
Properties
Object { get; } : object
ObjectId { get; } : Guid
Methods
UserTask(object obj, Guid objId)
Storage
Class
IStorage
Properties
Users { get; } : User[]
Methods
DoAction(string className, string methodName, IList&lt;object&gt; parameters) : void
GetCollectionByClassName(string className) : List&lt;object&gt;
GetUserByName(string userName) : User
ListMyDevTasks(Developer dev) : List&lt;DevTask&gt;
ListMyProjects(Manager mgr) : List&lt;Project&gt;
ListMyTestTasks(Developer dev) : List&lt;TestTask&gt;
ListProjects() : List&lt;Project&gt;
ListTasks() : List&lt;BaseTask&gt;
ListUsers() : List&lt;User&gt;
Load() : void
Save() : void
Storage()