**Dokumentation für Program.cs**

Program.cs dient als Einstiegspunkt für die C# Anwendung "VirtuSphere". Dieses Modul ist für das Initialisieren der Anwendung zuständig, einschließlich des Anzeigens des Anmeldebildschirms und des Hauptfensters der Anwendung. Es handhabt auch die Benutzeranmeldung und die Überprüfung des Verbindungstyps.

**Abhängigkeiten**

* **Newtonsoft.Json.Linq**: Wird verwendet, um mit JSON-Daten zu arbeiten, die von verschiedenen APIs der Anwendung benötigt werden.
* **System.Windows.Forms**: Ermöglicht das Erstellen von grafischen Benutzeroberflächen für die Anwendung.
* **VirtuSphere.apiService**: Enthält Dienste zur Interaktion mit der API.
* **VirtuSphere.FMmain**: Bezieht sich auf das Hauptformular der Anwendung, das nach der Anmeldung angezeigt wird.

**Funktionen und Methoden**

* **Main()**:
  + **Typ**: async Task
  + **Beschreibung**: Die Hauptmethode der Anwendung, die asynchron ausgeführt wird, um die Benutzeroberfläche und andere asynchrone Operationen zu verwalten. Sie ist der Startpunkt der Anwendung.
  + **Ablauf**:
    1. **Visual Styles aktivieren**: Stellt sicher, dass die Anwendung aktuelle Windows-Visuals verwendet.
    2. **Text Rendering einstellen**: Legt fest, dass die Anwendung kompatible Textdarstellungsstandards verwendet.
    3. **LoginForm-Instanzierung**: Erzeugt eine Instanz des LoginForm, um die Benutzeranmeldung zu verwalten.
    4. **DialogResult-Überprüfung**: Zeigt das LoginForm an und überprüft das Ergebnis. Wenn das Ergebnis nicht DialogResult.OK ist, wird die Anwendung beendet.
    5. **Benutzername speichern**: Speichert den Benutzernamen aus dem LoginForm zur weiteren Verwendung.
    6. **FMmain-Instanzierung**: Erzeugt eine Instanz von FMmain, dem Hauptfenster der Anwendung, mit Token und API-URL aus dem LoginForm.
    7. **Verbindungstyp überprüfen**: Überprüft, ob die Verbindung als verschlüsselt markiert ist, und aktualisiert das Label entsprechend.
    8. **Hauptanwendung ausführen**: Startet die Hauptanwendung mit Application.Run(mainForm).

**Nutzungsszenario**

Das Skript ist der erste Ausführungspunkt, wenn die Anwendung gestartet wird. Es verantwortet das Initialisieren der Anwendung, indem zuerst das Login-Formular geladen und nach erfolgreicher Authentifizierung das Hauptformular angezeigt wird. Die Anwendung überprüft aktiv, ob der Benutzer die Verbindung als "verschlüsselt" markiert hat und stellt diese Information im Hauptfenster dar.

Diese Dokumentation sollte Ihnen eine klare Vorstellung über die Funktionsweise und den Aufbau der Program.cs Datei in Ihrer C#-Anwendung geben. Sie hilft Entwicklern und neuen Benutzern, die Arbeitsweise und den Ablauf beim Start der Anwendung zu verstehen.

**Dokumentation für Form1.cs**

Form1.cs ist eine umfangreiche Datei für die Hauptform (FMmain) der Anwendung "VirtuSphere", die zahlreiche Funktionen zur Verwaltung virtueller Maschinen (VMs), Missionsdaten und Netzwerkressourcen bietet.

**Abhängigkeiten**

* **System**: Grundlegende Funktionalitäten für das Betriebssystem und die Umgebung.
* **System.Data**: Interaktion mit Datenstrukturen.
* **System.Drawing**: Bearbeitung und Verwaltung von Grafiken.
* **System.IO**: Dateioperationen.
* **System.Windows.Forms**: Grafische Benutzeroberfläche der Anwendung.
* **VirtuSphere.apiService**: API-Services für die Kommunikation mit Backend-Diensten.

**Funktionen und Methoden**

* **InitializeAsync()**: Asynchrone Initialisierung der Form, die Missionsdaten, Paketinformationen und VMs aus der API lädt und die Benutzeroberfläche entsprechend aktualisiert.
* **btn\_loadVMsfromDB()**: Lädt VMs aus der Datenbank basierend auf der ausgewählten Mission und zeigt sie an. Verwaltet auch Zustandsänderungen in der VM-Liste und bietet die Möglichkeit, ungespeicherte Änderungen zu speichern.
* **btnAddClick()**: Fügt eine neue VM zur Liste hinzu, nachdem die Eingabedaten validiert wurden.
* **btnEditClick()**: Bearbeitet eine ausgewählte VM und aktualisiert ihre Daten in der Anzeige.
* **btnDeleteClick()**: Entfernt eine ausgewählte VM nach einer Bestätigung durch den Benutzer.
* **btnCSVExportClick() und btnCSVImportClick()**: Exportiert die angezeigten Daten zu einer CSV-Datei bzw. importiert Daten aus einer CSV-Datei in die Anwendung.
* **CountdownTimer\_Tick()**: Verwaltet einen Countdown-Timer und aktualisiert die Benutzeroberfläche entsprechend. Warnt den Benutzer, wenn das API-Token abgelaufen ist.
* **LoadVMsIntoListView()**: Lädt VMs in die ListView-Komponente der Benutzeroberfläche.
* **DisableInputFields() und EnableInputFields()**: Aktiviert oder deaktiviert Eingabefelder basierend auf dem Zustand der Anwendung (z.B. wenn eine Mission ausgewählt ist).
* **UpdateListView()**: Aktualisiert die Liste der angezeigten VMs, um Änderungen in den VM-Daten widerzuspiegeln.
* **GetSelectedPackages()**: Ruft die vom Benutzer ausgewählten Pakete ab und ordnet diese den VMs zu.

**Nutzungsszenario**

Die Hauptform dient als zentrale Schnittstelle für die Verwaltung von virtuellen Maschinen, Netzwerkkonfigurationen und Missionsdaten. Sie bietet umfangreiche Funktionalitäten zum Hinzufügen, Bearbeiten und Löschen von VMs, zur Verwaltung von Softwarepaketen und zur Durchführung von Systemexporten und -importen. Sie bietet auch eine dynamische Anpassung der Benutzeroberfläche basierend auf dem aktuellen Zustand der Anwendung, wie z.B. der Auswahl bestimmter Missionsdaten oder VMs.

Diese Dokumentation beschreibt die Schlüsselfunktionen und Methoden, die in der Datei Form1.cs implementiert sind, und bietet eine Grundlage für das Verständnis und die weitere Entwicklung der Anwendung VirtuSphere.

**Dokumentation für vmeditForm.cs**

vmeditForm.cs enthält die Implementierung eines Formulars zur Bearbeitung von Details einer virtuellen Maschine (VM) in der Anwendung VirtuSphere. Es bietet interaktive Schnittstellen zur Änderung von VM-Eigenschaften, Netzwerkeinstellungen und Festplattenkonfigurationen.

**Abhängigkeiten**

* **System**: Basisfunktionalität für Windows Forms und andere Systemaktionen.
* **System.Collections.Generic**: Verwendung von generischen Kollektionen wie List.
* **System.Drawing**: Bereitstellung von Grafikfunktionen.
* **System.Linq**: Unterstützt das Durchsuchen von Datenstrukturen.
* **System.Threading.Tasks**: Unterstützung für asynchrone Operationen.
* **System.Windows.Forms**: Grundlage für das GUI-Design und die Interaktion.
* **VirtuSphere.apiService**: API-Services, die zur Interaktion mit Backend-Diensten verwendet werden.

**Hauptfunktionen und Methoden**

* **vmeditForm (Konstruktor)**: Initialisiert die Form und lädt die übergebene VM in die Formularelemente.
* **UpdateVMFromFormFields**: Aktualisiert die Eigenschaften der ausgewählten VM basierend auf den Formulardaten.
* **LoadVMToFormFields**: Lädt Daten einer VM in die Formularelemente zur Ansicht oder Bearbeitung.
* **GetSelectedPackages**: Ruft die ausgewählten Pakete ab, die der VM zugeordnet sind.
* **MarkSelectedPackagesInListBox**: Markiert die Pakete in der ListBox, die der VM zugeordnet sind.
* **AddInterfaceToListBox**: Fügt neue Netzwerkinterfaces zur VM hinzu.
* **ClearInterfaceDetails**: Löscht die Detailfelder des Interface-Bereichs im Formular.
* **btnSave\_Click**: Speichert die Änderungen und aktualisiert die VM-Daten.
* **btnClose, btnAddInterface, btnDeleteInterface**: Steuern das Schließen des Formulars, das Hinzufügen neuer Interfaces und das Löschen bestehender Interfaces.
* **TextBox\_TextChanged**: Behandelt Ereignisse für Textänderungen und steuert die Formularlogik basierend auf Eingaben des Benutzers.

**Nutzungsszenario**

vmeditForm wird aufgerufen, wenn ein Benutzer eine bestehende VM zur Bearbeitung auswählt. Es ermöglicht die detaillierte Bearbeitung von VM-Eigenschaften wie Name, Hostname, Betriebssystem, RAM, CPU und Speicher. Zusätzlich können Netzwerkeinstellungen wie IP-Adresse, Subnetz, Gateway und DNS konfiguriert werden. Die Form ermöglicht auch die Verwaltung von Festplattenkonfigurationen und das Hinzufügen oder Entfernen von Netzwerkinterfaces.

Diese Form spielt eine zentrale Rolle in der Verwaltung der VMs innerhalb der VirtuSphere-Anwendung, indem sie umfassende Bearbeitungsfunktionen bereitstellt, die direkt auf das Backend und die Datenhaltung einwirken.

**Dokumentation für MissionDetails.cs**

MissionDetails.cs stellt ein Formular bereit, das zur Anzeige und Bearbeitung der Details einer bestimmten Mission innerhalb der VirtuSphere-Anwendung verwendet wird. Es erlaubt die Modifikation von Missionseigenschaften wie Name, Notizen, Netzwerk-Konfigurationen und mehr.

**Abhängigkeiten**

* **System**: Basisfunktionalitäten und Datenstrukturen für die Anwendung.
* **System.ComponentModel, System.Data, System.Drawing**: Für die GUI-Gestaltung und Datenmanipulation.
* **System.Linq**: Wird genutzt, um effizient durch Datenstrukturen zu navigieren.
* **System.Windows.Forms**: Hauptbibliothek für die Erstellung von Formularen und deren Elementen.
* **VirtuSphere.apiService**: API-Services, die für Backend-Interaktionen genutzt werden.

**Hauptfunktionen und Methoden**

* **MissionDetails (Konstruktor)**: Initialisiert die Instanz der Form und lädt die übergebene Mission in die Formularelemente.
* **fillForms**: Lädt die Daten der spezifischen Mission in die GUI-Elemente zur Anzeige oder Bearbeitung.
* **btn\_save\_Click**: Übernimmt die im Formular eingegebenen Änderungen und speichert diese zurück in die Datenstruktur der Anwendung.
* **btn\_close\_Click**: Schließt das Formular ohne Speichern von Änderungen.

**Nutzungsszenario**

Die Klasse MissionDetails wird typischerweise aufgerufen, wenn ein Benutzer die Details einer Mission bearbeiten möchte. Sie ermöglicht es dem Benutzer:

* Missionsbezogene Informationen wie Namen, Notizen, und Netzwerk-Einstellungen zu bearbeiten.
* Neue Einstellungen für Datastorage und Datacenter festzulegen, was besonders wichtig für die Konfiguration der Missionsumgebung ist.
* Die Kopie von Missionen zu verwalten, um bestehende Konfigurationen als Vorlagen für neue Missionen zu verwenden.
* Eine Überprüfung durchzuführen, ob alle VMs innerhalb der Mission aktualisiert werden sollen, falls sich kritische Netzwerkeinstellungen ändern.

Die Schnittstelle ist so gestaltet, dass sie sowohl Informationsanzeige als auch interaktive Bearbeitung in einer kohärenten und benutzerfreundlichen Weise unterstützt. Die Anwendung nutzt die apiService-Klasse, um Daten persistent zu speichern oder zu modifizieren, was die Integrität und Aktualität der Missionsdaten sicherstellt.

Die ApiService.cs-Klasse ist eine zentrale Komponente der VirtuSphere-Anwendung, die für alle HTTP-basierten Interaktionen mit einer externen API zuständig ist. Die Klasse nutzt das .NET Framework’s HttpClient zur Ausführung von Webanfragen und behandelt eine Vielzahl von Aufgaben, die von der Authentifizierung und Session-Management bis hin zu spezifischen Geschäftslogiken wie dem Verwalten von virtuellen Maschinen (VMs), Betriebssystemen, Missionen und VLAN-Konfigurationen reichen.

**Schlüsselkomponenten der ApiService-Klasse:**

**HttpClient**

* Die Instanz von HttpClient wird verwendet, um Netzwerkanfragen zu senden und zu empfangen. Dies umfasst das Senden von GET, POST, PUT und DELETE Anfragen je nach erforderlicher Aktion (z.B. Abrufen oder Aktualisieren von Daten).

**Authentifizierung und API-Endpunkte**

* **apiToken** und **apiUrl** sind zentrale Variablen, die für die Authentifizierung bei der API und für die Konstruktion der Anfrage-URLs verwendet werden.
* Methoden prüfen regelmäßig den Status des API-Tokens, um sicherzustellen, dass die Nutzer-Session gültig bleibt.

**Hauptmethoden und ihre Funktionen**

**Authentifizierung**

* **IsValidLogin**: Diese Methode validiert Benutzeranmeldedaten, indem sie eine POST-Anfrage an den /api/login.php Endpunkt sendet. Sie setzt auch entsprechende Sicherheitsprotokolle, wenn TLS verwendet wird.

**CRUD-Operationen**

* **Create**, **Read**, **Update**, **Delete** Funktionen für verschiedene Ressourcen wie OS (CreateOS, UpdateOS, RemoveOS), VLANs (CreateVLAN, UpdateVLAN, RemoveVLAN), VMs und Missionen. Diese Methoden konstruieren die jeweiligen API-Anfragen und verarbeiten die Antworten.

**Datenabruf und Management**

* **GetPackages**, **GetMissions**, **GetOS**, **GetVLANs**, **GetVMs**: Diese Methoden senden GET-Anfragen an die API und deserialisieren die Antwort in entsprechende Objekte oder Listen von Objekten. Sie behandeln auch Fehlerfälle, wie z.B. abgelaufene API-Tokens oder ungültige Antworten.

**Fehlerbehandlung**

* Jede Methode ist mit einem umfangreichen Fehlerbehandlungsmechanismus ausgestattet, der Fehler wie Netzwerkprobleme, ungültige Antworten oder Probleme bei der JSON-Deserialisierung abfängt. Fehler führen zu Benutzerdialogen, die den Benutzer über Probleme informieren, oder zu Einträgen in einem Fehlerprotokoll.

**Beispiel: GetMissions-Methode**

Die GetMissions-Methode illustriert, wie die ApiService-Klasse verwendet wird, um Missionsdaten abzurufen:

csharp

Code kopieren

public async Task<List<MissionItem>> GetMissions()

{

string requestUri = $"http://{apiUrl}/access.php?action=getMissions&token={apiToken}";

var response = await \_httpClient.GetAsync(requestUri);

if (response.IsSuccessStatusCode)

{

var responseContent = await response.Content.ReadAsStringAsync();

try

{

var missionsList = JsonConvert.DeserializeObject<List<MissionItem>>(responseContent);

return missionsList;

}

catch (JsonException e)

{

// Fehlerhandling bei ungültigem JSON

}

}

return null; // Bei Fehlschlag oder "Access Forbidden"

}

Diese Methode ruft eine Liste von Missionen von der API ab und konvertiert die JSON-Antwort in eine Liste von MissionItem-Objekten, wobei sie sich um verschiedene Fehler kümmert, die während des Prozesses auftreten können.

**Nutzung im Kontext der Anwendung**

Die ApiService-Klasse ermöglicht es Entwicklern, sämtliche Netzwerkoperationen zentral zu verwalten und fördert damit die Wiederverwendbarkeit und Modularität des Codes. Sie wird überall dort aufgerufen, wo die Anwendung Daten vom Server benötigt oder Daten an diesen senden muss, und trägt so zur Entkopplung der Benutzeroberfläche von der Netzwerklogik bei.

Die AnsibleForm.cs-Klasse ist Teil einer umfangreichen Windows Forms-Anwendung, die als Schnittstelle zur Verwaltung von Konfigurationen und Bereitstellungen über Ansible dient. Die Klasse ermöglicht es den Benutzern, virtuelle Maschinen (VMs) zu konfigurieren und zu verwalten, indem sie SSH für Fernzugriffe und Ansible-Playbooks für automatisierte Task-Ausführungen verwendet.

**Schlüsselkomponenten und Funktionen der AnsibleForm-Klasse:**

**Daten- und Pfadmanagement**

* **PathTmp** und **ProjectPathTmp**: Temporäre Pfade für die Speicherung von Dateien und Skripten, die während der Sitzung verwendet werden.
* **missionName**, **ssh\_hostname**, **esxi\_hostname**: Verschiedene Konfigurationsparameter, die zur Identifikation und Verwaltung der Zielserver verwendet werden.

**UI-Elemente und Event Handler**

* **comboPlaybooks**: Eine Dropdown-Liste, die es Benutzern ermöglicht, aus verschiedenen vordefinierten Ansible-Playbooks zu wählen.
* **listFiles**: Eine Liste, die alle relevanten Konfigurationsdateien anzeigt, die im temporären Projektverzeichnis gespeichert sind.
* **txtAnsible**: Ein Textfeld zur Anzeige und Bearbeitung von Dateiinhalten.
* **checkYamlFile**: Eine Methode, die die Validität von YAML-Dateien überprüft und Feedback im UI gibt.

**SSH und Ansible Integration**

* **btnDeploy**: Ein Button, der den Bereitstellungsprozess über SSH startet. Diese Methode setzt voraus, dass der Benutzer entweder über einen SSH-Schlüssel oder über ein Passwort verfügt. Sie führt auch eine Überprüfung und gegebenenfalls eine Hinzufügung des öffentlichen Schlüssels zum Zielserver durch.
* **generateConfigs** und **reloadListFiles**: Methoden, die notwendig sind, um Konfigurationsdateien zu erstellen und die Liste der Dateien im UI zu aktualisieren.

**Datei- und Netzwerkoperationen**

* **loadConfig**: Lädt den Inhalt einer ausgewählten Datei zur Ansicht oder Bearbeitung.
* **btnSave\_Click**: Speichert Änderungen, die im txtAnsible-Textfeld gemacht wurden, zurück in die entsprechende Datei.
* **btn\_importMacDB\_Click**: Importiert MAC-Adressen von VMs von einem SSH-Server, was zeigt, dass die Anwendung auch in der Lage ist, Informationen vom Server abzurufen und zu verarbeiten.

**Beispiel: Die btnDeploy Methode**

Die btnDeploy-Methode zeigt, wie die Anwendung SSH verwendet, um Skripte auf einem Remote-Server auszuführen. Sie initialisiert eine SSH-Verbindung, prüft, ob ein SSH-Schlüssel vorhanden ist, und führt dann eine Reihe von Befehlen auf dem Remote-Server aus. Hier ist ein Auszug aus dieser Methode:

csharp

Code kopieren

var sshConnector = new SshConnector();

string privateKeyPath = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile) + "\\.ssh\\id\_rsa";

string publicKeyPath = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile) + "\\.ssh\\id\_rsa.pub";

if (!File.Exists(privateKeyPath)) {

// Generiere SSH-Schlüssel und füge ihn zum Remote-Server hinzu

publicKey = sshConnector.GenerateSSHKey(privateKeyPath);

string checkAndAddPublicKeyCommand = $"grep -q -F '{publicKey}' ~/.ssh/authorized\_keys || echo '{publicKey}' >> ~/.ssh/authorized\_keys";

// Führe den Befehl aus, um den Schlüssel hinzuzufügen

}

// Weitere Befehle ausführen...

**Nutzung im Kontext der Anwendung**

Die AnsibleForm-Klasse stellt eine komplexe Integration von Netzwerkmanagement, Dateioperationen und Benutzeroberflächeninteraktion dar. Sie ermöglicht es dem Benutzer, VMs effizient zu verwalten und Bereitstellungen über eine grafische Schnittstelle zu steuern, wobei die Automatisierung durch Ansible im Hintergrund abläuft. Dies reduziert Fehler, spart Zeit und verbessert die Effizienz bei der Verwaltung von IT-Infrastrukturen.

Die SshConnector-Klasse in C# stellt eine zentrale Schnittstelle für SSH-Operationen dar und nutzt die Renci.SshNet-Bibliothek. Diese Klasse ist speziell darauf ausgerichtet, SSH-Schlüsselpaare zu generieren, SSH-Befehle auszuführen und die SSH-Verbindung zu prüfen. Hier sind die wichtigsten Methoden und ihre Funktionen beschrieben:

**Methoden und ihre Funktionalitäten:**

**1. GenerateSSHKey**

* **Zweck**: Generiert ein RSA-Schlüsselpaar (öffentlichen und privaten Schlüssel) unter Verwendung des ssh-keygen-Befehls, der üblicherweise in UNIX-basierten Systemen vorhanden ist.
* **Parameter**: privateKeyPath - Der Pfad, an dem der private Schlüssel gespeichert wird. Der öffentliche Schlüssel wird automatisch am gleichen Ort mit der Erweiterung .pub gespeichert.
* **Rückgabe**: Gibt den Inhalt des öffentlichen Schlüssels als String zurück oder null im Fehlerfall.
* **Besonderheiten**: Die Methode nutzt ProcessStartInfo und Process für die Ausführung des ssh-keygen-Befehls und handhabt das Lesen des öffentlichen Schlüssels aus der Datei.

**2. ExecuteCommands**

* **Zweck**: Führt eine Liste von SSH-Befehlen auf einem entfernten Server aus.
* **Parameter**:
  + host: Die Adresse des SSH-Servers.
  + port: Der Port, auf dem der SSH-Server läuft.
  + username: Der Benutzername für den SSH-Login.
  + password: Das Passwort für den SSH-Login.
  + commands: Eine Liste von Strings, die die auszuführenden Befehle enthält.
* **Rückgabe**: Eine Liste von Strings, die die Ausgaben der Befehle enthalten. Bei einer fehlgeschlagenen Verbindung oder einem Fehler während der Ausführung wird eine Fehlermeldung zurückgegeben.
* **Besonderheiten**: Die Methode stellt die Verbindung her, führt jeden Befehl aus, sammelt die Ausgabe und trennt die Verbindung. Sie behandelt auch Fehler durch try-catch-Blöcke.

**3. CheckSshConnection**

* **Zweck**: Überprüft, ob eine SSH-Verbindung zu einem gegebenen Host möglich ist.
* **Parameter**:
  + host: Die Adresse des SSH-Servers.
  + port: Der Port, auf dem der SSH-Server läuft.
  + username: Der Benutzername für den SSH-Login.
  + password: Das Passwort für den SSH-Login.
* **Rückgabe**: true, wenn die Verbindung erfolgreich war, sonst false.
* **Besonderheiten**: Die Methode versucht eine Verbindung herzustellen und gibt true zurück, wenn die Verbindung erfolgreich ist. Bei jeglichen Fehlern während des Verbindungsversuchs wird false zurückgegeben.

**Beispiel für die Verwendung der SshConnector-Klasse:**

csharp

Code kopieren

SshConnector sshConnector = new SshConnector();

string privateKeyPath = "path/to/private/key";

string publicKey = sshConnector.GenerateSSHKey(privateKeyPath);

if (publicKey != null)

{

Console.WriteLine("SSH key generated successfully.");

}

else

{

Console.WriteLine("Failed to generate SSH key.");

}

List<string> commands = new List<string> { "uname -a", "whoami" };

List<string> output = sshConnector.ExecuteCommands("example.com", 22, "username", "password", commands);

foreach (string line in output)

{

Console.WriteLine(line);

}

bool isConnected = SshConnector.CheckSshConnection("example.com", 22, "username", "password");

Console.WriteLine(isConnected ? "Connection successful." : "Connection failed.");

Die SshConnector-Klasse bietet somit eine robuste und wiederverwendbare Schnittstelle für verschiedene SSH-bezogene Operationen innerhalb von .NET-Anwendungen, was besonders nützlich für Automatisierungs- und Fernwartungsszenarien ist.

Die Klasse VMManager.cs enthält Definitionen für verschiedene Klassen, die in einem typischen Virtual Machine Management Szenario verwendet werden könnten, wie VM, Interface, Disk und Package. Hier sind die Rollen und Funktionalitäten jeder dieser Klassen ausführlich beschrieben:

**1. VM (Virtual Machine)**

* **Attribute:**
  + ID, Mission ID, VM Name, Hostname, Domain, Betriebssystem, Speicher, Festplatte, CPU, Datastore, Datacenter, Guest ID, Ersteller, Status, Erstellt am, Modifiziert am, Notizen, MECM ID, ob aktualisiert, Interfaces, Pakete und Disks.
* **Methoden:**
  + DeepClone(): Erstellt eine tiefe Kopie einer VM-Instanz, einschließlich aller zugehörigen Interfaces, Packages, und Disks. Dies ist nützlich, um Änderungen an einer VM durchzuführen, ohne die Originalinstanz zu verändern.

**2. Interface**

* **Attribute:**
  + ID, VM ID, IP-Adresse, Subnetz, Gateway, DNS1, DNS2, VLAN, MAC-Adresse, Modus (z.B. DHCP oder statisch), Typ des Interfaces und ob es sich um ein Management-Interface handelt.
* **Methoden:**
  + Clone(): Erstellt eine flache Kopie eines Interface-Objekts. Nützlich, um schnell eine Kopie zu erstellen, ohne Referenzen zu kopieren.
* **Eigenschaften:**
  + DisplayText: Gibt einen beschreibenden Text für das Interface zurück, abhängig davon, ob es ein Management-Interface ist oder welche Konfiguration es verwendet.

**3. Disk**

* **Attribute:**
  + ID, VM ID, Name der Festplatte, Größe in Gigabyte, und Typ (z.B. SSD oder HDD).
* **Methoden:**
  + ToString(): Überschreibt die Standard ToString()-Methode, um eine lesbare Darstellung einer Disk zu bieten.
  + Clone(): Ermöglicht das Erstellen einer flachen Kopie einer Disk.

**4. Package**

* **Attribute:**
  + ID, Name des Pakets, Version und Status.
* **Methoden:**
  + Clone(): Bietet eine einfache Möglichkeit, eine Kopie eines Paket-Objekts zu erstellen.

**Nutzungsbeispiel:**

In einem hypothetischen Szenario, in dem VMManager zum Verwalten von VMs verwendet wird, könnten Sie Instanzen dieser Objekte erstellen, verwalten und manipulieren. Beispielhaft könnte der Code zum Klonen einer VM und zum Anzeigen ihrer Details wie folgt aussehen:

csharp

Code kopieren

VMManager manager = new VMManager();

VM vm = new VM()

{

vm\_name = "VM1",

vm\_hostname = "vm1.example.com",

vm\_domain = "example.com",

vm\_os = "Windows Server 2019",

vm\_ram = "16GB",

vm\_cpu = "4",

vm\_disk = "100GB",

vm\_datastore = "MainStore",

vm\_datacenter = "DataCenter1",

vm\_creator = "Admin",

vm\_notes = "Test VM",

interfaces = new List<Interface>()

{

new Interface() { ip = "192.168.1.100", subnet = "255.255.255.0", vlan = "VLAN1", mode = "Static" }

},

Disks = new List<Disk>()

{

new Disk() { disk\_name = "C Drive", disk\_size = 100, disk\_type = "SSD" }

}

};

VM clonedVM = vm.DeepClone();

Console.WriteLine($"Cloned VM Name: {clonedVM.vm\_name}");

In diesem Beispiel wird eine VM instanziiert, geklont und die Details der geklonten VM ausgegeben. Dieser Ansatz ermöglicht es, VM-Objekte effektiv und sicher zu verwalten und zu manipulieren, insbesondere in Umgebungen, in denen Änderungen häufig vorkommen und Rückgängig-Machbarkeit erforderlich ist.