Inhaltsverzeichnis

NetLogWin – Kompaktes Logging mit Benutzerinteraktion 2
Einleitung 2
NetLogWin Architekturübersicht 2
Pakete und Verantwortlichkeiten 2
Architekturprinzipien 2
Ablauf eines Logvorgangs 3
Installation & Einbindung 3
Abhängigkeiten 3
DLLs einbinden 3
Konfiguration (rein optional, direkt im Code) 3
FAQ – Häufig gestellte Fragen 4
Warum nicht einfach Console.WriteLine oder MessageBox.Show? 4
Muss ich zwingend ein Anzeige-Modul verwenden? 4
Kann ich Logging abschalten oder nur Debug-Meldungen erfassen? 4
Ist NetLogWin auch für große Projekte geeignet? 4
Was passiert bei Logging in Hintergrundthreads? 4
Sind weitere Userinteraktion als (J/N) möglich? 4
Wie funktioniert die automatische Logdatei-Rotation? 4
Sind Einstellungen auch über XML- oder JSON-Dateien möglich? 5
Geplant für eine vielleicht kommende Version 5
Vorteile der geplanten Erweiterung 5
Abwägung: Flexibilität vs. Minimalismus 5
Sicherheit bei geplanten Erweiterungen 5
Abbildungen / UML 6
UML der DLL`s, Klassen und Interface 6
Beispiel-Screen einer Consolen-App 7

NetLogWin – Kompaktes Logging mit Benutzerinteraktion

Einleitung

NetLogWin ist ein modulares Logging-Framework für .NET Framework 4.8 (und optional .NET 8+), das sich auf das Wesentliche konzentriert: **strukturierte Protokollierung** in Dateien und **einfache Benutzerinteraktion**, sowohl in **Konsolen-** als auch in **Windows-Anwendungen**.

Im Gegensatz zu vielen überladenen Logging-Frameworks erlaubt NetLogWin:

- Logdateien schnell, gezielt und mehrstufig zu schreiben (Info, Warnung, Fehler, Erfolg etc.).
- Rückfragen an den Benutzer zu stellen (z. B. Ja/Nein/Abbrechen) – **direkt im Logging-Aufruf.**
- Ausgabeziele (Konsole, GUI) einfach austauschbar zu halten, über klare Schnittstellen.

NetLogWin Architekturübersicht

NetLogWin ist ein schlankes, leistungsfähiges Logging-Framework für .NET 4.8 und .NET 8 Windows-Anwendungen. Es kombiniert Dateilogging mit direkter Benutzerinteraktion – das hebt es von großen Frameworks ab.

Pakete und Verantwortlichkeiten

Paketname	Typ	Beschreibung
HostApp_exe	Executable	Die Anwendungs-Startklasse (MainApplication) und Initialisierer (LogInitializer).
NetLogger48_dll	Kern-DLL	Enthält die Hauptlogging-Klassen: LogWriter, LogCore, LogFile-Sinks und die Display-Policy.
LogEnum48_dll	Enum-DLL	Gemeinsame Enums (LogLevel, UserResponse, ResponseButtons, LogFileRotation) für maximale Entkopplung.
IODisplay48Win_dll	Anzeige-DLL	Windows GUI-basierte Implementierung des Anzeige-Plugins (Dialoge, MessageBox, etc.).
IODisplay48Con_dll	Anzeige-DLL	Konsolenbasierte Implementierung des Anzeige-Plugins (Farbausgaben, Benutzereingaben in der Konsole).

Architekturprinzipien

- **Modularität:**
  Klare Trennung der Kern-Logging-Logik (`NetLogger48_dll`) von Anzeige-Implementierungen (`IODisplay48Win_dll/IODisplay48Con_dll`).
- **Entkopplung:**
  Gemeinsame Datentypen in `LogEnum48_dll` minimieren Abhängigkeiten zwischen DLLs.
- **Flexibilität:**
  Austauschbare Anzeige-Plugins ermöglichen Nutzung in verschiedensten Anwendungstypen.
- **Kombination Logging & Userinteraktion:**
  Einzigartig ist die Integration von Benutzerabfragen in den Logging-Prozess in einem einzigen Aufruf.
- **Leichtgewicht:**
  Kompakt und übersichtlich, ohne unnötigen Ballast – ideal für kleine bis mittlere Anwendungen.

Ablauf eines Logvorgangs

1. Die Anwendung ruft `LogWriter` auf, z.B. `LogWriter.Information("...")`.
2. `LogWriter` übergibt die Nachricht an `LogCore`.
3. `LogCore` schreibt je nach LogLevel in Logdateien (`FileLogSink`, `FileDebugLogSink`).
4. Über die konfigurierte `ILogDisplayPolicy` wird eine Anzeige-Callback-Methode (z.B. `IODisplayServiceWin.ShowMessage`) aufgerufen.
5. Benutzerinteraktion (z.B. Frage mit Ja/Nein) wird direkt verarbeitet und liefert Rückgabewerte.
6. Ergebnis wird an die Anwendung zurückgegeben.

Diese Dokumentation gibt Entwicklern eine klare Orientierung, wie sie NetLogWin einsetzen, erweitern und pflegen können.

Installation & Einbindung

Abhängigkeiten

- **.NET Framework 4.8** (Optionale Adaptionen für .NET 6/8 sind vorhanden)

DLLs einbinden

1. Referenziere die folgenden DLLs in deinem Projekt:
   - `NetLogger48.dll` – enthält den Logging-Core
   - `LogEnum48.dll` – zentrale Enums für Logging und Anzeige
   - Eines der folgenden Anzeige-Module:
     - `IODisplay48Con.dll` (für Konsolenanwendungen)
     - `IODisplay48Win.dll` (für Windows-Formulare)
2. Initialisiere den Logger in deinem Programmstart: (*minmale Anweisungen*)

```
Public Shared Sub InitializeLogging(displayMessageCallback As
DisplayExecutionCallback, Optional diagnosticWriterCallback As
DiagnosticLogWriterCallback = Nothing)
' === ERFORDERLICH ===
' Zuweisung der Logik-Policy (Anzeige) Kein diagnosticWriterCallback übergeben
LogCore.ActiveDisplayPolicy = New DefaultLogDisplayPolicy(
New DisplayExecutionCallback(AddressOf IODisplayServiceCon.ShowMessage))
End Sub
```

3. Nutze die Logging-Funktionen:

```
LogWriter.Information("Programm gestartet")
Dim answer As UserResponse = LogWriter.Question("Backup löschen?",
ResponseButtons.YesNo)
```

Konfiguration (rein optional, direkt im Code)

Folgende Optionen sind per Code konfigurierbar – **es sind keine XML- oder app.config-Dateien nötig**:

- Mindest-Loglevel für Dateiausgabe (`LogCore.MinimumFileLogLevel`)
- Logfile-Rotation (z. B. stündlich, täglich etc.) über `LogCore.LogFileRotation`
- Maximalgröße einzelner Logdateien (`FileLogSink.MaxLogFileSizeMB`)
- Steuerung der Anzeige- und Interaktionslogik über eigene `ILogDisplayPolicy`-Implementierungen

Die Konfiguration ist **nicht erforderlich** – du kannst mit wenigen Zeilen sofort produktiv loggen.

FAQ – Häufig gestellte Fragen

Warum nicht einfach `Console.WriteLine` oder `MessageBox.Show`?

Weil NetLogWin beides **vereinheitlicht**, strukturiert loggt und dennoch Rückfragen an den Benutzer erlaubt – im selben Aufruf.
Beispiel:

```
Dim result = LogWriter.Question("Datei wirklich löschen?", ResponseButtons.YesNo)
If result = UserResponse.Yes Then …
```

Muss ich zwingend ein Anzeige-Modul verwenden?

Nein. Ein Anzeige-Modul (`IODisplay..`) ist **nur erforderlich**, wenn du **Benutzerinteraktion** wie Fragen, Warnungen oder Hinweise auf der Konsole oder im GUI anzeigen möchtest.
Für reines File-Logging (z. B. Fehlerprotokollierung) genügt `NetLogger48.dll` allein.

Wenn du eigene Anzeige-Logik nutzen willst, kannst du eine eigene Policy schreiben, die das Interface `ILogDisplayPolicy` implementiert.

Kann ich Logging abschalten oder nur Debug-Meldungen erfassen?

Ja. Die interne `LogCore`-Klasse steuert das Verhalten über LogLevel-Filter und Rotation.

Ist NetLogWin auch für große Projekte geeignet?

NetLogWin ist für **kompakte Tools und schlanke Desktop-Apps** gedacht – es ersetzt keine Enterprise-Logging-Systeme, aber lässt sich leicht erweitern.

Was passiert bei Logging in Hintergrundthreads?

Alle LogWriter-Aufrufe sind **thread-safe**. Benutzerinteraktionen erfolgen im Aufrufkontext (kein versteckter `Invoke` oder UI-Magie).

Sind weitere Userinteraktion als (J/N) möglich?

Ja. Die `LogEnum48.dll` definiert verschiedene Antwortmodelle über den `ResponseButtons`-Enum – z. B. `(Ja/Nein/Abbrechen)`, `(OK/Wiederholen)` u. v. m.
Eine vollständige Übersicht findest du in der Sandcastle-Dokumentation.

Wie funktioniert die automatische Logdatei-Rotation?

Die Log-Rotation wird über den Enum `LogFileRotation` in der `LogEnum48.dll` gesteuert.
Folgende Modi stehen zur Verfügung:

- `SingleFile` Es wird immer dieselbe Datei beschrieben.
- `Hourly` Jede Stunde wird eine neue Logdatei erzeugt.
- `Daily` Täglich eine neue Datei (Standard).
- `Weekly` Wöchentliche Rotation.
- `Monthly` Eine Datei pro Kalendermonat.

Die Auswahl erfolgt direkt im Code – ganz ohne XML oder AppSettings.
Der erzeugte Dateiname enthält automatisch ein Zeitpräfix (`_H`, `_D`, `_W`, `_M`) für eine klare Zuordnung.

Wichtig:
Unabhängig von der gewählten Rotation greift zusätzlich die `MaxLogFileSize`-Begrenzung.
Wird die maximale Größe überschritten, wird ebenfalls automatisch eine neue Datei begonnen – mit hochgezähltem Suffix (_01, _02, …).

Sind Einstellungen auch über XML- oder JSON-Dateien möglich?

In der aktuellen Version: nein.
NetLogWin wurde bewusst so konzipiert, dass **alle Konfigurationswerte direkt im Quellcode gesetzt** werden – inklusive LogLevel, Rotation, Anzeigeverhalten (Policies) und Dateigrößen.

Das Ziel:
Ein **schlankes Logging-Framework** für .NET-Anwendungen mit maximaler Kontrolle **ohne externe Abhängigkeiten**.
Kein Aufwand mit Konfigurationsparsern, Dateiüberwachung, Validierungslogik oder Pfadauflösung – alles liegt in der Hand des Entwicklers.

Geplant für eine vielleicht kommende Version

Ein optionales Konfigurations-Plugin (z. B. auf Basis von XML oder JSON) ist für zukünftige Releases vorgesehen.
Wichtig: **Die Nutzung bleibt dann vollständig freiwillig.**
Entwickler können selbst entscheiden, ob sie weiterhin rein per Code arbeiten oder Konfigurationsdateien einbinden möchten.

Vorteile der geplanten Erweiterung

- Ideal für größere oder modulare Projekte
- Ermöglicht einfache Anpassung von Logverhalten **ohne Rebuild**

Abwägung: Flexibilität vs. Minimalismus

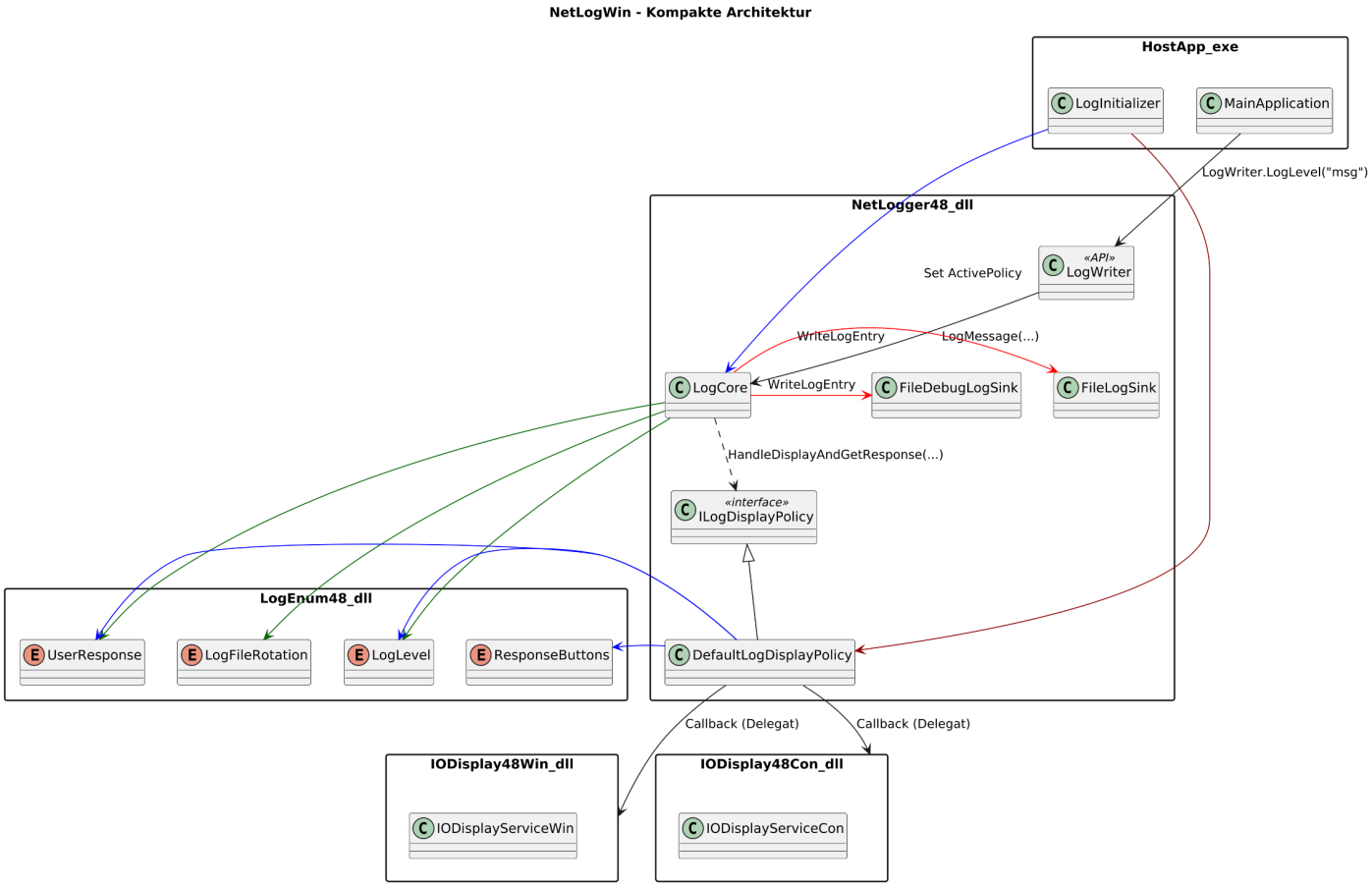
Aspekt	Aktuelle Lösung (Codebasiert)	Zukünftige Option (Konfigurierbar)
Setup-Komplexität	Sehr gering	Etwas höher (Parser + Datei nötig)
Abhängigkeiten	Keine	Möglicherweise nötig (Plugin)
Änderbarkeit ohne Rebuild	Nein	Ja
Wartbarkeit in größeren Projekten	Manuell im Code	Strukturierter über externe Dateien
Sicherheit gegenüber den User	Sehr Gut	Eher gering

Sicherheit bei geplanten Erweiterungen

Durch Konfigurationsdateien können Nutzer – gewollt oder versehentlich – Einstellungen manipulieren (z. B. Debug-Log aktivieren, Pfade ändern oder Anzeigeverhalten beeinflussen).
In der rein **codebasierten Variante** ist das ausgeschlossen, was den Schutz der Anwendung deutlich erhöht – besonders in sicherheitsrelevanten Bereichen oder geschlossenen Systemen.

Abbildungen / UML

UML der DLL`s, Klassen und Interface



Beispiel-Screen einer Consolen-App