CWF Engine basiert auf der Originalimplementierung von Wexflow <https://github.com/aelassas/Wexflow>. Lizenziert unter MIT License.

**Wie kann man dieses Verwenden bzw. Installieren?**

Der CWF Workflow wird als Quellcode ausgeliefert. Dieser wurde mit Visual Studio 2017 erstellt und ist als .Net Core Solution abgelegt. Die CWF Workflow Engine kann in jeder beliebiger .Net Core Anwendung verwendet werden. Weiters kann die CWF Workflow Engine auch in Service Fabric Anwendungen integriert werden, da diese komplett von außen konfigurierbar gemacht worden ist.

**Wie lässt sich der CWF Workflow konfigurieren?**

Der CWF Workflow lässt sich über die Klasse CWFEngine instanziieren. Die Klasse ist über ihren Konstruktor vollständig parametrisierbar. Eine CWFEngine Instanz ist vollständig instanzierbar sobald die folgenden Variablen festgelegt sind.

|  |  |
| --- | --- |
| WorkflowsFolder | Pfad mit den Workflow Definitionen |
| XsdFilePath | Pfad der Datei welche die aktuelle Workflow Struktur definiert. |
| ActivitiesPath | Unterverzeichnis mit den Activities |
| StateMachinePath | In diesem Unterverzeichnis befinden sich die State Dlls. |

**Für was wird der Workflow verwendet? Wie funktioniert er?**

Mit dem CWF Workflow lassen sich algorithmische Abläufe abbilden. Es gibt zum einen Activities die Funktionalität in Form von Funktionsblöcken die als Dll implementiert sind verarbeiten. Zum anderen gibt es Transitionen die die logischen Übergänge definieren.

Jede Transition hat einen Startindex und einen Zielindex. Weiters kann für jede Transition eine Condition, eine so genannte Guard, festgelegt werden. Diese kann Werte aus dem internen Zustandsobjekt abfragen und als Bedingung verarbeiten.

Die Workflow Definition wird für jeden Workflow separat in einer XML-Datei mit folgender Syntax:

<Workflow xmlns="urn:wexflow-schema" id="3" name="HanoiFlowchart" description="A demo workflow">

<Settings>

<Setting name="launchType" value="startup" /> <!-- startup|trigger|periodic -->

<Setting name="enabled" value="true" /> <!-- true|false -->

<Setting name="statemachinetype" value="HanoiWorkflowState" />

</Settings>

<Activities>

<Activity id="99" name="SetupHanoiGameActivity" description="evenBetter" enabled="true">

<Setting name="ParameterType" value="HanoiSetupConfiguration" />

<Setting name="NumberDisks" value="7" />

</Activity>

<Activity id="1" name="CheckHanoiNotFinishedActivity" description="DrScott" enabled="true"/>

<Activity id="2" name="RunHanoiRoundActivity" description="Timewarp" enabled="true" />

<Activity id="4" name="PostHanoiStateActivity" description="ScienceFiction" enabled="true" />

</Activities>

<Transitions>

<Transition id="1" from="-1" to="99"/>

<Transition id="2" from="99" to="2"/>

<Transition id="3" from="2" to="4" />

<!-- Transition to loop -->

<Transition id="4" from="4" to="2" condition="!x.IsFinished" />

</Transitions>

</Workflow>



**Wie wird der Workflow verwendet?**

Der Workflow kann zum einen als Quellcode in eine Solution eingebunden werden. Es kann auch eine DLL daraus gemacht werden und in ein Projekt integriert werden. Weiters wird beim kompilieren der Solution auch ein Nuget Packet erstellt.

**Abhängigkeiten**

Es sind keine direkten Abhängigkeiten vorhanden. Es reicht, wenn auf der auszuführenden Maschine das .Net SDK installiert ist.

**How to install**

CWF Engine is based on Wexflow's original implementation <https://github.com/aelassas/Wexflow>. Licensed under MIT License.

**How to configure the Workflow**

The CWF workflow can be instantiated using the class CWFEngine. The class can be fully parameterized via its constructor. A CWFEngine instance is fully instantiated when the following variables are set.

|  |  |
| --- | --- |
| WorkflowsFolder | Path with the workflow definitions |
| XsdFilePath | Path of the file which defines the current workflow structure. |
| ActivitiesPath | Subdirectory with the activities |
| StateMachinePath | In this subdirectory the State Dlls are located. |

**What is the workflow for? How it works?**

The CWF workflow can be used to map algorithmic processes. On the one hand, there are activities that process functionality in the form of function blocks that are implemented as DLLs. On the other hand, there are transitions that define the logical transitions.  
Each transition has a start index and a target index. Furthermore, a condition, a so-called guard, can be defined for each transition. This can query values from the internal state object and process them as a condition.  
The workflow definition is separate for each workflow in an XML file with the following syntax:

<Workflow xmlns="urn:wexflow-schema" id="3" name="HanoiFlowchart" description="A demo workflow">

<Settings>

<Setting name="launchType" value="startup" /> <!-- startup|trigger|periodic -->

<Setting name="enabled" value="true" /> <!-- true|false -->

<Setting name="statemachinetype" value="HanoiWorkflowState" />

</Settings>

<Activities>

<Activity id="99" name="SetupHanoiGameActivity" description="evenBetter" enabled="true">

<Setting name="ParameterType" value="HanoiSetupConfiguration" />

<Setting name="NumberDisks" value="7" />

</Activity>

<Activity id="1" name="CheckHanoiNotFinishedActivity" description="DrScott" enabled="true"/>

<Activity id="2" name="RunHanoiRoundActivity" description="Timewarp" enabled="true" />

<Activity id="4" name="PostHanoiStateActivity" description="ScienceFiction" enabled="true" />

</Activities>

<Transitions>

<Transition id="1" from="-1" to="99"/>

<Transition id="2" from="99" to="2"/>

<Transition id="3" from="2" to="4" />

<!-- Transition to loop -->

<Transition id="4" from="4" to="2" condition="!x.IsFinished" />

</Transitions>

</Workflow>



**How to use the workflow.**

The workflow can be integrated into a solution as source code. It can also be a DLL built out of it and integrated into a project. Furthermore, when compiling the solution also a Nuget Packet is created.

**Dependencies/requirements**

There are no direct dependencies. It is sufficient if the .Net SDK is installed on the machine to be executed.