# Bau einer Activity basierend auf CWF Engine

## Beschreibung

Projekte die mit dem CWF Workflow erstellt werden, bestehen aus einer oder mehreren Aktivitäten die mit Hilfe einer C# Erweiterungsbibliothek erstellt werden. Jede dieser Aktivities implementiert eine eigene Run Methode.

## Parameterliste

Die Parameterliste sieht folgendermaßen aus:

public override void Run(System.Threading.CancellationToken token, object state, object parameterDto = null)

|  |  |
| --- | --- |
| System.Threading.CancellationToken token | Token zum Canceln von laufendem Workflow. |
| object state | Das StateToken |
| object parameterDto = null | Parameter Dto mit den Parameter Werten der Activity |

## Parameter Dto

Das Parameter Dto wird für jede Aktivität generiert. Dafür werden die Setting Tags im Settings Bereich interpretiert:

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

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

Im Beispielcode wird Beispielsweise ein Settingsobject vom Type HanoiSetupConfiguration generiert. In diesem befindet sich ein Property von den Datentypen integer und diesem wird der Wert 7 zugewiesen. In diesem Fall wird der Wert einfach zum Initialisieren des Workflows verwendent.

## Einbindung einer Aktivität in eine Workflow Xml

<Workflow xmlns="urn:cwf-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>

## Implementierung einer CWF-Aktivität

Die Aktivität liegt in Form einer C# Dll vor und wird in das CWFEngine/Activities Verzeichnis kopiert.

## Prinzipieller Aufbau der CWFEngine

Die CWFEngine hat eine fixe Ordnerstruktur in den sämtlichen Dateien wie Activities und States eingeordnet werden. Jeder Workflow besteht aus einen oder mehreren Aktivitäten die durchlaufen werden. Für jeden Workflow wird auch ein Workflowstate instanziiert. Dieser ist für jede Aktivität zugänglich und damit lassen sich interne Daten speichern.

## Kurzanleitung

Um eine Aktivität zu erstellen wird Visual Studio in der Version 2017 benötigt. Das Dll Projekt muss als .NET Core 2.1 Projekt kompiliert werden. Das Ergebnis des Projektes ist eine .NET Core Dll.

Nachfolgend ist der Quellcode einer Aktivität in kompilierbarer Form dargestellt:

//-----------------------------------------------------------------------

// <copyright file="PostHanoiStateToActiveMQActivity.cs" company="Breanos GmbH">

// Copyright Notice:

// DAIPAN - This file, program or part of a program is considered part of the DAIPAN framework by Breanos GmbH for Industrie 4.0

// Published in 2018 by Gerhard Eder gerhard.eder@breanos.com and Achim Bernhard achim.bernhard@breanos.com

// To the extent possible under law, the publishers have dedicated all copyright and related and neighboring rights to this software to the public domain worldwide. This software is distributed without any warranty.

// You should have received a copy of the CC0 Public Domain Dedication along with this software. If not, see <http://creativecommons.org/publicdomain/zero/1.0/>.

// <date>Tuesday, October 30, 2018 2:54:51 PM</date>

// </copyright>

//-----------------------------------------------------------------------

using System;

using System.Xml.Linq;

using BreanosConnectors;

using CWF.Core;

using CWF.Core.ExecutionGraph;

using HanoiLibrary;

using NLog;

using static CWF.Core.Parser;

namespace CWF.Tasks.PostHanoiStateToActiveMQActivity

{

public class PostHanoiStateToActiveMQActivity : CWF.Core.StatefulActivity<HanoiWorkflowState>

{

private static NLog.Logger logger = LogManager.GetCurrentClassLogger();

public PostHanoiStateToActiveMQActivity(ActivityMemento activityMememnto) : base(activityMememnto)

{

}

public override void Run(System.Threading.CancellationToken token, object state, object parameterDto = null)

{

if (!token.IsCancellationRequested)

{

var s = state as HanoiLibrary.HanoiWorkflowState;

StateToken = s;

StateToken.PropertyChanged += StateToken\_PropertyChanged;

StateToken.DiskBaseWidth = 30;

string serialized = SerializationHelper.Pack(s);

Core.Logger.InfoFormat($"PostHanoiStateToActiveMQActivity sending...{serialized}");

Workflow.Engine.Amqc.SendAsync(serialized, Workflow.Engine.TopicName, typeof(string).AssemblyQualifiedName);

}

else

{

Core.Logger.InfoFormat($"PostHanoiStateToActiveMQActivity has been canceled by user");

}

FinishActivity();

}

private void StateToken\_PropertyChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)

{

}

}

}

Besonders darauf hingewiesen sei z.B. dass die Aktivität von : **CWF.Core.StatefulActivity<HanoiWorkflowState>** erbt. HanoiWorkflowState ist in diesem Fall eine Klasse die den internen Zustand des Workflows repräsentiert.

In der Methode Run sei auf den Umgang mit dem Cancelation Token hingewiesen. Diese muss vom Programmierer in der Ausführung des Codes berücksichtigt werden um einen Workflow jederzeit durch die Workflow Engine terminierbar zu machen.

Wichtig ist auch dass am Ende der Run Methode **FinishActivity** aufgerufen wird. Es gibt auch ein FinishActivity, dass mit einen Rückgabeparameter true/false aufgerufen wird.