## Betriebskonzept

Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

### Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 07.06.2012 | 1.1 | Erste Version des Dokuments | CH |
| 11.06.2012 | 1.2 | Review Installationsanleitung | DT |
| 12.06.2012 | 1.3 | Review | DT |

### Installationsdokumentation

#### Dokumentation für Administrator

Damit die Applikation verwendet werden kann wird folgendes benötigt:

* Kompilierte Version des Projektes
* .NET, Kinect Runtime
* Internetverbindung

Die kompilierten Dateien müssen als erstes in den gewünschten Zielordner kopiert werden.   
Weiter können verschiedene Plug-ins zur Applikation hinzugefügt werden. Damit dies möglich ist, muss die Ordnerstruktur nach der Anleitung im Kapitel I.1.8.3 Plug-ins aufgebaut werden.

#### Dokumentation für Entwickler

Damit die Applikation weiterentwickeln werden kann, muss ein SVN Checkout auf dem gesamten SVP Repository durchgeführt werden. Dazu wird ein SVN Tool wie z.B. TortoiseSVN benötigt (siehe Tools TODO).

Zudem werden folgende Werkzeuge benötigt:

* Visual Studio 10
* Expression Blend
* ReSharper
* Kinect for Windows SDK
* NuGet

Die Konfiguration der Applikation findet in der app.config-Datei statt. Wie die Konfiguration angepasst werden kann, wird im Kapitel I.1.8.4 Konfigurationsdatei erläutert.   
Weiter können der Applikation Plug-ins hinzugefügt werden, dies wird im nachfolgenden Kapitel (I.1.8.3 Plug-ins) aufgezeigt.

#### Plug-ins

Im app.config-File kann der Pfad zum Ordner, welcher die Plug-ins enthalten soll, konfiguriert werden. Wie dieser Pfad gesetzt und angepasst werden kann, wird im Unterkapitel 0 Abbildung 42 - app.config, Sektion Unity, Padding

Pfad zu den Plug-ins erläutert.

Für jede Plug-in-Applikation, die man auf der Videowall anzeigen möchte, muss ein entsprechender Unterordner im Plug-ins Ordner (siehe 0 Abbildung 42 - app.config, Sektion Unity, Padding

Pfad zu den Plug-insAbbildung 42 - app.config, Sektion Unity, Padding

Pfad) erstellt werden, in welchem dann die Files des Plug-ins gespeichert werden. Ein solcher Ordner darf nie leer sein, da sonst beim Starten der Videowall-Applikation eine entsprechende Fehlermeldung angezeigt wird.   
Bestimmte Plug-in-Applikationen benötigen zusätzliche Dateien wie beispielsweise Bilder. Diese müssen in einem Unterordner mit dem Namen „Files“ im Ordner des Plug-ins abgelegt werden. Fehlt dieser Ordner, so wird eine entsprechende Fehlermeldung beim Starten der Hauptapplikation angezeigt.

#### Konfigurationsdatei

Die Einstellungen, welche in der Konfigurationsdatei getätigt werden können, sind nachfolgend aufgezeigt.

##### Konfigurationssektionen

Im Abschnitt configSections können verschiedene Konfigurationssektionen definiert werden.

  <configSections>

    <section name="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection, Microsoft.Practices.Unity.Configuration"/>

  </configSections>

Abbildung - app.config, Konfigurationssektionen

##### Unity

In der Konfigurationssektion Unity werden die Namespaces angegeben, in denen die Klassen gesucht werden sollen, welche über Unity instanziiert werden sollen. Zu den Namespaces muss weiter angegeben werden, in welchem Assembly diese sich befinden.

    <namespace name="VideoWall.Data.Kinect.Implementation"/>

    <namespace name="VideoWall.Data.Kinect.Interfaces"/>

    <assembly name="VideoWall.Data"/>

    <namespace name="VideoWall.ServiceModels.Apps.Implementation"/>

    <namespace name="VideoWall.ServiceModels.Apps.Interfaces"/>

    <namespace name="VideoWall.ServiceModels.DemoMode.Implementation"/>

    <namespace name="VideoWall.ServiceModels.DemoMode.Interfaces"/>

    <namespace name="VideoWall.ServiceModels.HandCursor.Implementation"/>

    <namespace name="VideoWall.ServiceModels.HandCursor.Interfaces"/>

    <namespace name="VideoWall.ServiceModels.Player.Implementation"/>

    <namespace name="VideoWall.ServiceModels.Player.Interfaces"/>

    <assembly name="VideoWall.ServiceModels"/>

    <namespace name="VideoWall.ViewModels"/>

    <namespace name="VideoWall.ServiceModels.Apps"/>

    <namespace name="VideoWall.ViewModels.Cursor"/>

    <namespace name="VideoWall.ViewModels.HitButton"/>

    <namespace name="VideoWall.ViewModels.Menu"/>

    <namespace name="VideoWall.ViewModels.Skeletons"/>

    <assembly name="VideoWall.ViewModels"/>

Abbildung 37 - app.config, Sektion Unity, Namespaces&Assemblies

###### Interfaces auf konkrete Klassen mappen

Weiter können in der Sektion Unity mithilfe des Tags <container> Interfaces auf konkrete Klassen gemappt werden. Dies wird wie folgt angegeben:

        <type type="IDemoModeService" mapTo="DemoModeService"/>

        <type type="IAppController" mapTo="AppController"/>

        <type type="IHandCursorPositionCalculator" mapTo="HandCursorPositionCalculator"/>

        <type type="IPlayer" mapTo="Player"/>

Abbildung - app.config, Sektion Unity, Mapping von Interfaces auf Klassen

###### Skelettdaten

Der *ISkeletonReader* dient dazu, die Skelettdaten zu Lesen.   
Der *KinectSkeletonReader* wird verwendet, wenn Kinect angeschlossen ist und die Videowall normal betrieben wird.   
Für Testzwecke werden der *FileSkeletonReader* und der *AutoPlayFileSkeletonReader*, welche mit Kinect aufgenommene Skelettdaten abspielen, angeboten. Der *FileSkeletonReader* spielt das File einmal, der *AutoPlayFileSleketonReader* spielt das File immer wieder von Neuem ab. Das abzuspielende File mit den Skelettdaten kann über den Typ *KinectReplayFile* definiert werden, was im Unterkapitel 0 Abbildung 45 - app.config, Sektion Unity, Singleton

KinectReplayFile erläutert wird.

        <!-- FileSkeletonReader -->

        <!-- AutoPlayFileSkeletonReader -->

        <!-- KinectSkeletonReader -->

        <type type="ISkeletonReader" mapTo="KinectSkeletonReader">

          <lifetime type="singleton"/>

        </type>

Abbildung 39 - app.config, Sektion Unity, Mapping für ISkeletonReader

###### Cursor

Ist der Kinect Sensor am Computer angeschlossen, so ist für das Mapping vorzugsweise das *KinectCursorViewModel* zu verwenden. Der Cursor, welcher in der Applikation als kleine Hand dargestellt wird, kann so durch Handbewegungen vor dem Kinect Sensor gesteuert werden. Ist das Mapping für den *ISkeletonReader* (siehe Unterkapitel I.1.8.4.2.2 Skelettdaten) auf *FileSkeletonReader* oder *AutoPlayFileSkeletonReader* gesetzt, so wird der Handcursor durch das Skelett im abzuspielenden File gesteuert.

Wird das Schlüsselwort *MouseCursorViewModel* verwendet, so lässt sich die Applikation mit der Maus bedienen. Trotzdem wird auch hier eine Hand angezeigt.

        <!-- KinectCursorViewModel -->

        <!-- MouseCursorViewModel -->

        <type type="ICursorViewModel" mapTo="MouseCursorViewModel">

          <lifetime type="singleton"/>

        </type>

Abbildung - app.config, Sektion Unity, Mapping für ICursorViewModel

##### Padding

Damit die Applikation angenehm mit Kinect zu bedienen ist, wurde einer Interaktionszone für die Hand definiert. Das Prinzip dieser Zone wird im Kapitel (TODO. Entwurf, Handtracking) genauer erläutert. Die rote Fläche in der Abbildung 41 - Interaktionszone ist auf den Arm des Nutzers ausgerichtet und stellt die eigentliche Interaktionszone dar. Die schwarz umrahmte Fläche stellt den Erkennungsbereich von Kinect dar. Das *RelativePadding* wird verwendet, um diese rote Zone zu definieren. Möchte man die Position und Grösse der Zone verändern, so kann dies durch Anpassungen an den Werten der aufgelisteten Schlüsselwörter *left*, *top*, *right* und *bottom* getan werden. Welchen Abstand diese Schlüsselwörter verändern ist in der Abbildung 41 - Interaktionszone ersichtlich.

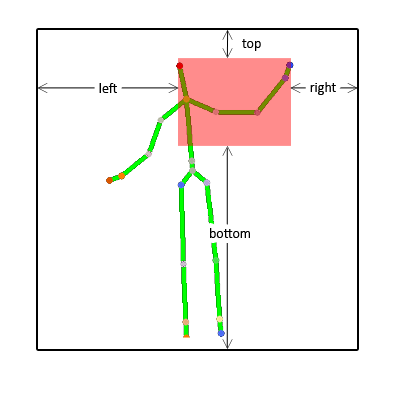


Abbildung 41 - Interaktionszone

        <type type="RelativePadding" mapTo="RelativePadding">

          <constructor>

            <param name="left" value="0.45"/>

            <param name="top" value="0.1"/>

            <param name="right" value="0.3"/>

            <param name="bottom" value="0.49"/>

          </constructor>

        </type>

Abbildung 42 - app.config, Sektion Unity, Padding

###### Pfad zu den Plug-ins

Die Applikation kann durch Plug-ins dynamisch erweitert werden. Diese müssen in einem bestimmten Ordner abgelegt werden. Der Parameter *extensionsDirectoryPath* muss auf den gewünschten Pfad gesetzt werden.

        <type type="ExtensionsConfig" mapTo="ExtensionsConfig">

          <constructor>

            <param name="extensionsDirectoryPath" value="../../../Extensions"/>

          </constructor>

        </type>

Abbildung 43 - app.config, Sektion Unity, Pfad zu den Plug-ins

###### Demomodus

Der Demomodus wird aktiv, wenn Kinect keinen Nutzer erkennt. Einstellungen für den Demomodus können im Bereich *IDemoModeConfig* vorgenommen werden.

        <type type="IDemoModeConfig" mapTo="DemoModeConfig">

          <constructor>

            <param name="backgroundColors">

              <array>

                <value value="#ff0065a3"/>

                <value value="#ff6e1c50"/>

                <value value="#ff548c86"/>

                <value value="#ff7b6951"/>

                <value value="#ff00738d"/>

                <value value="#ffbabd5d"/>

              </array>

            </param>

            <param name="skeletonCheckTimeSpan" value="00:00:00.01"/>

            <param name="changeAppTimeSpan" value="00:00:05"/>

            <param name="fromActiveToDemoModeTimeSpan" value="00:00:15"/>

            <param name="countdownTimeSpan" value="00:00:04.999"/>

            <param name="skeletonTrackingTimeoutTimeSpan" value="00:00:00.5"/>

          </constructor>

        </type>

Abbildung 44 - app.config, Sektion Unity, Demomodus

Als Werte für den Parameter *backgroundColors* können Farben für den Hintergrund des Demomodus angegeben werden.

Des Weiteren können die Werte der verschiedenen Timer angepasst werden.   
Der *skeletonCheckTimeSpan* gibt an, nach wie vielen Millisekunden erneut geprüft wird, ob gerade ein Nutzer erkannt wurde.  
Über den *changeAppTimeSpan* kann die Zeitspanne bis zum Wechsel des Teaser-Textes und der Hintergrundfarbe im aktiven Demomodus definiert werden.  
Wenn die Applikation aktiv ist aber kein Skelett mehr erkennt, wechselt sie nach Ablauf des *fromActiveToDemoModeTimeSpan* in den Demomodus.   
Wird im Demomodus ein Nutzer erkannt, so wird ein Countdownzähler angezeigt, welcher den Übergang in den Interaktionsmodus andeutet. Die Länge dieses Zählers kann über den *countdownTimeSpan* verändert werden.   
Nachdem ein Nutzer von Kinect erkannt wurde, wird immer wieder geprüft, ob der Nutzer immer noch erkannt wird. Die Zeitspanne bis zur nächsten Überprüfung wird über den *skeletonTrackingTimeoutTimeSpan* angegeben.

###### Singelton

Von bestimmten Klassen darf es immer nur eine Instanz geben, welche dann an mehreren Orten verwendet werden kann.   
Die Klasse Player muss ein Singleton sein: Das Skelett des Kinect Players wird zum einen beim Übergang vom Demomodus in den Interaktionsmodus und schliesslich im Interaktionsmodus selbst angezeigt. Die Problematik mit Kinect ist, dass das Gerät nur von einer Instanz angesteuert werden kann. Bestünden nun zwei verschiedene Skelette, die das Kinect ansteuern und Daten vom Gerät verlangen, würde das in einem Fehler enden.   
Im Falle des MenuViewModels darf nur eine Instanz existieren, welche alle Plug-ins im Menu aufzeigt und den Inhalte des aktuell angewählten auf der Videowall anzeigt.

        <type type="Player" mapTo="Player">

          <lifetime type="singleton"/>

        </type>

        <type type="MenuViewModel" mapTo="MenuViewModel">

          <lifetime type="singleton"/>

        </type>

Abbildung 45 - app.config, Sektion Unity, Singleton

###### KinectReplayFile

Wird im app.config beim Mapping auf den ISkeletonReader (siehe I.1.8.4.2.2 Skelettdaten) der *FileSkeletonReader* oder der *AutoPlayFileSkeletonReader* verwendet, so muss ein *KinectReplayFile* angegeben werden. Dieses beinhaltet Daten zu Skelettbewegungen. Der Pfad zu dieser Datei kann an folgender Stelle beim Schlüsselwort *path* angegeben werden:

 <type type="KinectReplayFile" mapTo="KinectReplayFile">

          <lifetime type="singleton"/>

          <constructor>

            <param name="path" value="..\..\..\..\..\..\kinect\_records\20120312\_lukas\\_1.replay"/>

          </constructor>

        </type>

Abbildung 46 - app.config, Sektion Unity, KinectReplayFile

Der Pfad zum Replay-File ist relativ anzugeben.

##### Runtime Version

Über das Schlüsselwort *runtime* können die Pfade zu den neuen Assembly-Versionen und deren Speicherort angegeben werden. Diese Angaben werden benötigt um Assembly-Versionskonflikte zu vermeiden.

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="Microsoft.Kinect" publicKeyToken="31BF3856AD364E35" culture="neutral"/>

<bindingRedirect oldVersion="1.0.0.0-1.5.0.0" newVersion="1.5.0.0"/>

</dependentAssembly>

</assemblyBinding>

</runtime>

Abbildung - app.config, Runtime