ApartmentScence

ApartmentScence.cs

```
//------------------------------------------------------------------------
//  This file is part of Microsoft Robotics Developer Studio Code Samples.
//
//  Copyright (C) Microsoft Corporation.  All rights reserved.
//
//  $File: ApartmentScene.cs $ $Revision: 5 $
//------------------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
using submgr = Microsoft.Dss.Services.SubscriptionManager;
using engine = Microsoft.Robotics.Simulation.Engine.Proxy;
using simulatedwebcam = Microsoft.Robotics.Services.Simulation.Sensors.SimulatedWebcam.Proxy;
using webcam = Microsoft.Robotics.Services.WebCam.Proxy;
using simulateddrive = Microsoft.Robotics.Services.Simulation.Drive.Proxy;
using drive = Microsoft.Robotics.Services.Drive.Proxy;

#region Using Statements
using Microsoft.Dss.Core;
using System.Drawing;
using Microsoft.Ccr.Adapters.WinForms;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using Microsoft.Robotics.Services.Samples;
using Microsoft.Robotics.Simulation.Engine;
#endregion

namespace Robotics.ApartmentScene
{
    [Contract(Contract.Identifier)]
    [DisplayName("(User) Apartment Scene")]
    [Description("Shows how to perform basic image processing on an image from the simulated webcam
service in an apartment environment")]
    class ApartmentSceneService : DsspServiceBase
    {
        /// <summary>
        /// Service state
        /// </summary>
```

```
46          [ServiceState]
47          ApartmentSceneState _state = new ApartmentSceneState();
48
49          /// <summary>
50          /// Main service port
51          /// </summary>
52          [ServicePort("/ApartmentScene", AllowMultipleInstances = true)]
53          ApartmentSceneOperations _mainPort = new ApartmentSceneOperations();
54
55          [SubscriptionManagerPartner]
56          submgr.SubscriptionManagerPort _submgrPort = new submgr.SubscriptionManagerPort();
57
58          #region Partners
59          /// <summary>
60          /// SimulationEngine partner
61          /// </summary>
62          [Partner("SimulationEngine", Contract = engine.Contract.Identifier, CreationPolicy =
63  PartnerCreationPolicy.UseExistingOrCreate)]
64          engine.SimulationEnginePort _simulationEnginePort = new engine.SimulationEnginePort();
65          engine.SimulationEnginePort _simulationEngineNotify = new engine.SimulationEnginePort();
66
67          /// <summary>
68          /// SimulatedWebcamService partner
69          /// </summary>
70          [Partner("SimulatedWebcamService", Contract = webcam.Contract.Identifier, CreationPolicy =
71  PartnerCreationPolicy.UsePartnerListEntry)]
72          webcam.WebCamOperations _simulatedWebcamServicePort = new webcam.WebCamOperations();
73
74          /// <summary>
75          /// SimulatedDifferentialDriveService partner
76          /// </summary>
77          [Partner("SimulatedDifferentialDriveService", Contract = drive.Contract.Identifier,
78  CreationPolicy = PartnerCreationPolicy.UsePartnerListEntry)]
79          drive.DriveOperations _simulatedDifferentialDriveServicePort = new drive.DriveOperations();
80
81          #endregion
82
83          /// <summary>
84          /// Service constructor
85          /// </summary>
86          public ApartmentSceneService(DsspServiceCreationPort creationPort)
87              : base(creationPort)
88          {
89          }
90
91          #region Data Members
92          Port<DateTime> _dateTimePort = new Port<DateTime>();
93
94          // used to display gradient we compute
95          ImageProcessingResultForm _imageProcessingForm;
96          #endregion
97
```

```
        /// <summary>
        /// Service start
        /// </summary>
        protected override void Start()
        {

            //
            // Add service specific initialization here
            //

            base.Start();

            #region Activate Timer
            Activate(Arbiter.ReceiveWithIterator(false, _dateTimePort, UpdateImage));
            TaskQueue.EnqueueTimer(TimeSpan.FromMilliseconds(60), _dateTimePort);
            #endregion

            #region Run Win Form
            WinFormsServicePort.Post(new RunForm(() =>
            {
                _imageProcessingForm = new ImageProcessingResultForm();
                _imageProcessingForm.Show();
                return _imageProcessingForm;
            }));
            #endregion
        }

        #region Update Image
        IEnumerator<ITask> UpdateImage(DateTime dateTime)
        {
            byte[] rgbData = null;
            Size size = new Size(0, 0);

            yield return Arbiter.Choice(_simulatedWebcamServicePort.QueryFrame(),
                success =>
                {
                    rgbData = success.Frame;
                    size = success.Size;
                },
                failure =>
                {
                    LogError(failure.ToException());
                });

            if (rgbData != null)
            {
                ComputeGradient(ref rgbData, size);
                UpdateBitmap(rgbData, size);
            }

            Activate(Arbiter.ReceiveWithIterator(false, _dateTimePort, UpdateImage));
            TaskQueue.EnqueueTimer(TimeSpan.FromMilliseconds(60), _dateTimePort);
```

```
150         }
151         #endregion
152
153         #region Additional Methods
154         private void UpdateBitmap(byte[] rgbData, Size size)
155         {
156             if (_imageProcessingForm == null)
157                 return;
158
159             #region Running code on the WinForms thread
160             WinFormsServicePort.Post(new FormInvoke(() =>
161                 {
162                     Bitmap bmp = _imageProcessingForm.ImageResultBitmap;
163                     CopyBytesToBitmap(rgbData, size.Width, size.Height, ref bmp);
164                     if (bmp != _imageProcessingForm.ImageResultBitmap)
165                     {
166                         _imageProcessingForm.UpdateForm(bmp);
167                     }
168                     _imageProcessingForm.Invalidate(true);
169                 }));
170             #endregion
171         }
172
173
174         #region Basic Image Processing Methods
175         private void ComputeGradient(ref byte[] rgbData, Size size)
176         {
177             byte[] gradient = new byte[rgbData.Length];
178             int[,] mask = new[,]
179                 {
180                     {+2, +1, 0},
181                     {+1, 0, -1},
182                     {0, -1, -2}
183                 };
184             const int filterSize = 3;
185             const int halfFilterSize = filterSize / 2;
186
187             //convolve use simple n^2 method, but this can easily be made 2n
188             for (int y = halfFilterSize; y < size.Height - halfFilterSize; y++)
189             {
190                 for (int x = halfFilterSize; x < size.Width - halfFilterSize; x++)
191                 {
192                     float result = 0;
193                     for (int yy = -halfFilterSize; yy <= halfFilterSize; ++yy)
194                     {
195                         int y0 = yy + y;
196                         for (int xx = -halfFilterSize; xx <= halfFilterSize; ++xx)
197                         {
198                             int x0 = xx + x;
199                             int k = mask[yy + halfFilterSize, xx + halfFilterSize];
200                             int i = 3 * (y0 * size.Width + x0);
201                             int r = rgbData[i];
```

```
                            int g = rgbData[i + 1];
                            int b = rgbData[i + 2];
                            result += k * (r + g + b) / (3.0f);
                        }
                    }
                    result /= 4.0f; // normalize by max value
                    //the "result*5" makes edges more visible in the image, but is not really necessary
                    //  (only nice for display purposes)
                    byte byteResult = Clamp(Math.Abs(result * 5.0f), 0.0f, 255.0f);
                    int idx = 3 * (y * size.Width + x);
                    gradient[idx] = byteResult;
                    gradient[idx + 1] = byteResult;
                    gradient[idx + 2] = byteResult;
                }
            }

            rgbData = gradient;
        }

        private byte Clamp(float x, float min, float max)
        {
            return (byte)Math.Min(Math.Max(min, x), max);
        }


        /// <summary>
        /// Updates a bitmap from a byte array
        /// </summary>
        /// <param name="srcData">Should be 32 or 24 bits per pixel (ARGB or RGB format)</param>
        /// <param name="srcDataWidth">Width of the image srcData represents</param>
        /// <param name="srcDataHeight">Height of the image srcData represents</param>
        /// <param name="destBitmap">Bitmap to copy to. Will be recreated if necessary to copy to the
array.</param>
        private void CopyBytesToBitmap(byte[] srcData, int srcDataWidth, int srcDataHeight, ref Bitmap
destBitmap)
        {
            int bytesPerPixel = srcData.Length / (srcDataWidth * srcDataHeight);
            if (destBitmap == null
                || destBitmap.Width != srcDataWidth
                || destBitmap.Height != srcDataHeight
                || (destBitmap.PixelFormat == PixelFormat.Format32bppArgb && bytesPerPixel == 3)
                || (destBitmap.PixelFormat == PixelFormat.Format32bppRgb && bytesPerPixel == 3)
                || (destBitmap.PixelFormat == PixelFormat.Format24bppRgb && bytesPerPixel == 4))
            {
                if (bytesPerPixel == 3)
                    destBitmap = new Bitmap(srcDataWidth, srcDataHeight, PixelFormat.Format24bppRgb);
                else
                    destBitmap = new Bitmap(srcDataWidth, srcDataHeight, PixelFormat.Format32bppRgb);
            }
            BitmapData bmpData = null;
            try
            {
```

```
                if (bytesPerPixel == 3)
                    bmpData = destBitmap.LockBits(new Rectangle(0, 0, srcDataWidth, srcDataHeight),
ImageLockMode.WriteOnly, PixelFormat.Format24bppRgb);
                else
                    bmpData = destBitmap.LockBits(new Rectangle(0, 0, srcDataWidth, srcDataHeight),
ImageLockMode.WriteOnly, PixelFormat.Format32bppRgb);

                Marshal.Copy(srcData, 0, bmpData.Scan0, srcData.Length);
                destBitmap.UnlockBits(bmpData);
            }
            catch (Exception)
            {
            }
        }
        #endregion
        #endregion

        /// <summary>
        /// Handles Subscribe messages
        /// </summary>
        /// <param name="subscribe">the subscribe request</param>
        [ServiceHandler]
        public void SubscribeHandler(Subscribe subscribe)
        {
            SubscribeHelper(_submgrPort, subscribe.Body, subscribe.ResponsePort);
        }

        /// <summary>
        /// Handles Drop Messages
        /// </summary>
        /// <param name="drop"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Teardown)]
        public IEnumerator<ITask> DropHandler(DsspDefaultDrop drop)
        {
            // cancel any active tasks that are still active
            TaskQueue.Dispose();

            DefaultDropHandler(drop);
            yield break;
        }
    }
}
```

ImageProcessingResultForm.cs

```
//------------------------------------------------------------------------
//  This file is part of Microsoft Robotics Developer Studio Code Samples.
//
//  Copyright (C) Microsoft Corporation.  All rights reserved.
//
//  $File: ImageProcessingResultForm.cs $ $Revision: 3 $
//------------------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Imaging;

namespace Microsoft.Robotics.Services.Samples
{
    public partial class ImageProcessingResultForm : Form
    {
        #region WinForm members
        public Bitmap ImageResultBitmap { get; set; }
        public PictureBox PictureBox { get { return _formPicture; } }
        public long TimeStamp { get; set; }
        #endregion

        public ImageProcessingResultForm()
        {
            InitializeComponent();

            #region Initialize WinForm members
            ImageResultBitmap = new Bitmap(_formPicture.Width, _formPicture.Height);
            _formPicture.Image = ImageResultBitmap;
            #endregion
        }

        #region UpdateForm
        internal void UpdateForm(Bitmap bmp)
        {
            ImageResultBitmap = bmp;
            this.Size = new Size(bmp.Width, bmp.Height);
            _formPicture.Size = new Size(bmp.Width, bmp.Height);
            _formPicture.Image = ImageResultBitmap;
            this.Invalidate(true);
        }
        #endregion
    }
}
```