Using RAGE Asset complying to the RAGE Architecture

Contents

Intro

TODO

1) Example implementations for Unity3D

Adding client asset complying with the rage asset architecture is a bit more work than just dropping a script but as most extra work (implementing interfaces on the Bridge) has to be done only once for multiple assets the game programmer should benefit.

Setup Project

Rage C# assets that are usable on multiple devices and development environments have two C# project files.

The first one is for building a .Net 3.5 assembly (for Unity3D) and contains all sources.

The second one is a portable assembly for use within Xamarin and uses linked sources from the first project.

By using linked sources both projects can be compiled together without code duplication and .Net version specific issues will surface immediately.

The rage assets need a reference to the Rage Asset Manager (also consists of two C# project files). The .Net 3.5 asset needs a reference to the .Net 3.5 Asset Manager and the portable assembly project needs a reference to the portable Asset Manager project.

Multiple assets can be compiled against the same Asset Manager using a single solution.

Solution structure:

Asset manager

Asset manager (portable)

Asset #1

Asset #1 (portable)

Asset #2

Asset #2 (portable)

Untity3D

Due to the fact that Unity3D frequently re-creates (or at best modifies) the solution file it uses, the assets can best be compiled outside Unity3D using Visual Studio. The resulting .net 3.5 assemblies van be dropped in the Unity3D asset folders and Unity3D will recognize them. If the Visual Studio debug symbol files (*.pdb) are converted into Mono debug symbol files (*.mdb) using a suitable version of pdb2mdb, debugging and single stepping into the assemblies using Unity3D is possible.

Xamarin

In Xamarin one can just make a reference to the portable asset assemblies. It is possible to use the .Net 3.5 assemblies as well but it tends to create problems with the platform specific parts of a Xamarin project. Therefor portable assemblies are preferred.

Integration in a Game

If a Rage Asset needs to access game engine, device or operating system functionality (like saving a file), it needs code to be written that implements a certain interface. The interfaces required by an asset need to be implemented on a plain class (called the bridge) that implements at least the IBridge interface.

These interfaces are defined within the Rage Asset or the Rage Asset Manager assembly. By defining the interfaces in the Rage Asset Manager assembly, they can be re-used by multiple assets and thus needs to implemented only once.

Most common is the IDataStorage interface that is used for saving, loading and managing files. It has simple methods that a game developer needs to implement in such way that it fits the game and the target devices. For IDataStorage the device depended part is where to store files so they can be retrieved when needed. The Game Developers should not care about the content of a file other than that it is plain text (or base64 encoded binary data).

An in memory storage could simply be using a Dictionary<string,string> class for storage.

Another interface commonly used is IWebRequest. This interface has a single method WebRequest that has all the parts of an http request (method, protocol, port, host, url, headers etc.) as parameters. These parameters can be used to perform a web request. Returns are the parts of the response.

Note the code is synchronous as .Net 3.5 and portable assemblies differ a lot (simple yield versions full blown task parallel support using async/await). Therefore, if needed, the game programmer can call parts or sequences of the asset's API using a suitable async pattern instead of the asset dictating a pattern.

Bridge objects can be attached to either the Asset Manager (such bridge is then available to all Assets) and/or to a specific assets.

Common Bridge interfaces

The three most common interfaces are IDataStorage, ILog and IWebServiceRequest.

IDataStorage Interface

IDataStorage can be easily implemented using File.IO methods like File.ReadAllText or using Dictionary<String,String> in which case it will be in-memory storage. The only 'tricky' part is getting the

correct base directory of where to save files on a particular device. Unity3D, Xamarin differ here on popular devices using iOS/Android.

Note: On Unity3D it's not a good idea to derive the bridge from MonoBehaviour as in that case the constructor will run on a different thread and access to certain API methods is prohibited (specifically getting the user data directory). Better is to add a MonoBehaviour property the bridge (this property can also be shared with IWebServiceRequest) and set it using a constructor taking a MonoBehaviour as parameter.

```
/*
 * Copyright 2016 Open University of the Netherlands
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * This project has received funding from the European Union's Horizon
 * 2020 research and innovation programme under grant agreement No 644187.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
namespace AssetPackage
{
    using System;

    /// <summary>
    /// Interface for data storage.
    /// </summary>
    public interface IDataStorage
    {
        #region Methods

        /// <summary>
        /// Deletes the given fileId.
        /// </summary>
        ///
        /// <param name="fileId"> The file identifier to delete. </param>
        ///
        /// <returns>
        /// true if it succeeds, false if it fails.
        /// </returns>
        Boolean Delete(String fileId);

        /// <summary>
        /// Check if exists the file with the given identifier.
        /// </summary>
        ///
        /// <param name="fileId"> The file identifier to delete. </param>
        ///
        /// <returns>
        /// true if it succeeds, false if it fails.
```

```
        /// </returns>
        Boolean Exists(String fileId);

        /// <summary>
        /// Gets the files.
        /// </summary>
        ///
        /// <remarks>
        /// A List&lt;String&gt; gave problems when compiled as PCL and added to a
        /// Xamarin Forms project containing iOS, Android and WinPhone subprojects.
        /// </remarks>
        ///
        /// <returns>
        /// An array of filenames.
        /// </returns>
        String[] Files();

        /// <summary>
        /// Loads the given file.
        /// </summary>
        ///
        /// <param name="fileId"> The file identifier to load. </param>
        ///
        /// <returns>
        /// A String with with the file contents.
        /// </returns>
        String Load(String fileId);

        /// <summary>
        /// Saves the given file.
        /// </summary>
        ///
        /// <param name="fileId">   The file identifier to delete. </param>
        /// <param name="fileData"> Information describing the file. </param>
        void Save(String fileId, String fileData);

        #endregion Methods
    }
}
```

ILog

The ILog interface is used by assets for diagnostic logging purposes. It's up to the game programmer what to do with it (depending on the Severity level).

Note: The BaseAsset class has two wrappers for this Interface, so no further code is necessary:

```
        public void Log(Severity severity, String format, params object[] args);
        public void Log(Severity severity, String msg);
```

Note: Filtering on severity level can be performed in the Bridge implementation of this interface.

```
/*
 * Copyright 2016 Open University of the Netherlands
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * This project has received funding from the European Union's Horizon
```

```
 * 2020 research and innovation programme under grant agreement No 644187.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
namespace AssetPackage
{
    using System;

    /// <summary>
    /// Interface for logger.
    /// </summary>
    public interface ILog
    {
        /// <summary>
        /// Executes the log operation.
        ///
        /// Implement this in Game Engine Code.
        /// </summary>
        ///
        /// <param name="severity"> The severity. </param>
        /// <param name="msg">      The message. </param>
        void Log(Severity severity, String msg);
    }

    /// <summary>
    /// Values that represent log severity.
    /// <br/>
    ///      See
    /// <a href="https://msdn.microsoft.com/en-
us/library/office/ff604025(v=office.14).aspx">Trace
    /// and Event Log Severity Levels</a>
    /// </summary>
    public enum Severity : int
    {
        /// <summary>
        /// An enum constant representing the critical option.
        /// </summary>
        Critical = 1,

        /// <summary>
        /// An enum constant representing the error option.
        /// </summary>
        Error = 2,

        /// <summary>
        /// An enum constant representing the warning option.
        /// </summary>
        Warning = 4,

        /// <summary>
        /// An enum constant representing the information option.
```

```
        /// </summary>
        Information = 8,

        /// <summary>
        /// An enum constant representing the verbose option.
        /// </summary>
        Verbose = 16
    }

    /// <summary>
    /// Values that represent log levels.
    /// </summary>
    public enum LogLevel : int
    {
        /// <summary>
        /// An enum constant representing the critical option.
        /// </summary>
        Critical = Severity.Critical,
        /// <summary>
        /// An enum constant representing the error option.
        /// </summary>
        Error = Critical | Severity.Error,
        /// <summary>
        /// An enum constant representing the warning option.
        /// </summary>
        Warn = Error | Severity.Warning,
        /// <summary>
        /// An enum constant representing the information option.
        /// </summary>
        Info = Warn | Severity.Information,
        /// <summary>
        /// An enum constant representing all option.
        /// </summary>
        All = Severity.Critical | Severity.Error | Severity.Warning | Severity.Information
| Severity.Verbose,
    }
}
```

IWebServiceRequest

IWebServiceRequest consists of two helper classes one (RequestSettings) for passing the WebRequest parameters and one (RequestResponse) for returning the WebResponse data and an Interface to be implemented on the Bridge.

Implementing is a matter of setting up a UnityWebRequest in Unity3D or a regular (Http)WebRequest for Xamarin to do the actual work.

Note: Unity3D's WWW class is not sufficient as it only allows a very limited set of http verbs (GET/POST). Therefore UnityWebRequest is recommend.

Note: Both UnityWebRequest and WebRequest do not allow certain http headers to be added as they are already present as property (content-type and accept-type).

Note: The interface expects the WebRequest to be made synchronously (but wrapper methods chaining some calls can be called async as a whole). Reason is a) the differences between Unity3D and Xamarin

(yield vs async/await) and b) let the game programmer decide how and when to use multi-threading instead.

```
/*
 * Copyright 2016 Open University of the Netherlands
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * This project has received funding from the European Union's Horizon
 * 2020 research and innovation programme under grant agreement No 644187.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
namespace AssetPackage
{
    using System;
    using System.Collections.Generic;

    /// <summary>
    /// Request Settings.
    /// </summary>
    public class RequestSetttings
    {
        /// <summary>
        /// The method.
        /// </summary>
        public string method;

        /// <summary>
        /// URI of the document.
        /// </summary>
        public Uri uri;

        /// <summary>
        /// The request headers.
        /// </summary>
        public Dictionary<String, String> requestHeaders;

        /// <summary>
        /// The body.
        /// </summary>
        public String body;

        /// <summary>
        /// The allowed responses.
        /// </summary>
        public List<int> allowedResponsCodes;

        /// <summary>
        /// Initializes a new instance of the AssetPackage.requestParameters
```

```
        /// class.
        /// </summary>
        public RequestSetttings()
        {
            method = "GET";
            requestHeaders = new Dictionary<String, String>();
            body = String.Empty;
            allowedResponsCodes = new List<int>();
            allowedResponsCodes.Add(200);
        }
    }

    /// <summary>
    /// Response results.
    /// </summary>
    public class RequestResponse : RequestSetttings
    {
        /// <summary>
        /// The response code.
        /// </summary>
        public int responseCode;

        /// <summary>
        /// Message describing the respons.
        /// </summary>
        public string responsMessage;

        /// <summary>
        /// The response headers.
        /// </summary>
        public Dictionary<String, String> responseHeaders;

        /// <summary>
        /// Initializes a new instance of the AssetPackage.RequestResponse class.
        /// </summary>
        public RequestResponse() : base()
        {
            responseCode = 0;
            responsMessage = String.Empty;

            responseHeaders = new Dictionary<String, String>();
        }

        /// <summary>
        /// Initializes a new instance of the AssetPackage.RequestResponse class.
        /// </summary>
        ///
        /// <remarks>
        /// The body is not copied as it will contain thee response body instead.
        /// </remarks>
        ///
        /// <param name="settings"> Options for controlling the operation. </param>
        public RequestResponse(RequestSetttings settings) : this()
        {
            method = settings.method;
            requestHeaders = settings.requestHeaders;
            uri = settings.uri;
            body = String.Empty;
```

```
            allowedResponsCodes = settings.allowedResponsCodes;
        }

        /// <summary>
        /// Gets a value indicating whether result is allowed.
        /// </summary>
        ///
        /// <value>
        /// true if result allowed, false if not.
        /// </value>
        public bool ResultAllowed
        {
            get
            {
                return allowedResponsCodes.Contains(responseCode);
            }
        }
    }

    /// <summary>
    /// Interface for web service request.
    /// </summary>
    ///
    /// <remarks>
    /// Implemented on a Bridge.
    /// </remarks>
    public interface IWebServiceRequest
    {
        /// <summary>
        /// Web service request.
        /// </summary>
        ///
        /// <param name="requestSettings">Options for controlling the operation. </param>
        ///
        /// <returns>
        /// A RequestResponse.
        /// </returns>
        void WebServiceRequest(RequestSetttings requestSettings, out RequestResponse
requestResponse);
    }
}
```

Sample Bridge Implementations

IWebServiceRequest for Unity3D

```
#region IWebServiceRequest implementation

// http://docs.unity3d.com/ScriptReference/Experimental.Networking.UnityWebRequest.html
//
// Note: We use a synchronous way to invoke a 'Coroutine'.
//
// The idea is that the Asset Method is invoked using a Coroutine and not these methodes
// themselves that are invoked by the asset. The reason is that inside the asset we need
// the answers of this method (like the Auth Token, or be sure the GameStorage structure has
// been restored before restoring data etc).
//
// Unity3D differ a lot from Xamarin in this (async) matter.
//
// The WWW class is not used as UCM used PUT and DElETE als http methods (WWW only supports GET and POST).
//
public void WebServiceRequest (RequestSetttings requestSettings, out RequestResponse requestResponse)
{
        requestResponse = new RequestResponse(requestSettings);
```

```
        InternalWebServiceRequest(requestSettings, requestResponse);

        // Debug.Log(String.Format("{0} - {1}",  requestResponse.responseCode, requestResponse.uri));
}

// 'Fake' Coroutine.
private Coroutine InternalWebServiceRequest(RequestSetttings requestSettings, RequestResponse requestResponse)
{
        UnityWebRequest request = new UnityWebRequest();
        // Debug.Log(requestSettings.uri);

        request.method = requestSettings.method;
        request.url = requestSettings.uri.ToString();

        foreach (KeyValuePair<String,String> kvp in requestSettings.requestHeaders) {
                request.SetRequestHeader(kvp.Key, kvp.Value);
        }

        if (!String.IsNullOrEmpty(requestSettings.body))
        {
                byte[] data = Encoding.UTF8.GetBytes(requestSettings.body);
                UploadHandlerRaw upHandler = new UploadHandlerRaw(data);
                upHandler.contentType = requestSettings.requestHeaders["Content-Type"];
request.uploadHandler = upHandler;
        }

        DownloadHandler dnHandler = new DownloadHandlerBuffer();
        request.downloadHandler = dnHandler;
        request.Send();

        while (!request.isDone) {
                // Nothing
        }
```

```
        if (request.isError) {
                // Debug.Log(request.error);
                requestResponse.responsMessage = request.error;

        } else {
                requestResponse.responseCode = (int)request.responseCode;
                requestResponse.responseHeaders=request.GetResponseHeaders();
                requestResponse.body=request.downloadHandler.text;

                // Unity does not have a responseMessage available.
                requestResponse.responsMessage = String.Empty;
        }

        return null;

}

#endregion
```

IWebServiceRequest for PCL/Xamarin/.Net 4.5+

Note: Defining the ASYNC symbol means using async/await for the webrequest inside the bridge code.
Note: Without ASYNC, the webrequest is made synchronously (but can be wrapped inside a game engine async wrapper).

```
#region IWebServiceRequest Members
#if ASYNC
      public void WebServiceRequest(RequestSetttings requestSettings, out RequestResponse requestReponse)
      {
         // Wrap the actual method in a Task. Neccesary because we cannot:
         // 1) Make this method async (out is not allowed)
         // 2) Return a Task<RequestResponse> as it breaks the interface (only void does not break it).
         //
         Task<RequestResponse> taskName = Task.Factory.StartNew<RequestResponse>(() =>
         {
            return WebServiceRequestAsync(requestSettings).Result;
         });
         requestReponse = taskName.Result;
      }

      /// <summary>
      /// Web service request.
      /// </summary>
      ///
      /// <param name="requestSettings"> Options for controlling the operation. </param>
      ///
      /// <returns>
      /// A RequestResponse.
      /// </returns>
      private async Task<RequestResponse> WebServiceRequestAsync(RequestSetttings requestSettings)
#else
      /// <summary>
      /// Web service request.
```

```
        /// </summary>
        ///
        /// <param name="requestSettings"> Options for controlling the operation. </param>
        /// <param name="requestResponse">The request response. </param>
        public void WebServiceRequest(RequestSetttings requestSettings, out RequestResponse requestResponse)
        {
            requestResponse = WebServiceRequest(requestSettings);
        }

        /// <summary>
        /// Web service request.
        /// </summary>
        ///
        /// <param name="requestSettings">Options for controlling the operation. </param>
        ///
        /// <returns>
        /// A RequestResponse.
        /// </returns>
        private RequestResponse WebServiceRequest(RequestSetttings requestSettings)
#endif
        {
            RequestResponse result = new RequestResponse(requestSettings);
            try
            {
                //! Might throw a silent System.IOException on .NET 3.5 (sync).
                HttpWebRequest request = (HttpWebRequest)WebRequest.Create(requestSettings.uri);
                request.Method = requestSettings.method;

                // Both Accept and Content-Type are not allowed as Headers in a HttpWebRequest.
                // They need to be assigned to a matching property instead.
                if (requestSettings.requestHeaders.ContainsKey("Accept"))
                {
```

```
            request.Accept = requestSettings.requestHeaders["Accept"];
        }

        if (!String.IsNullOrEmpty(requestSettings.body))
        {
            byte[] data = Encoding.UTF8.GetBytes(requestSettings.body);
            if (requestSettings.requestHeaders.ContainsKey("Content-Type"))
            {
                request.ContentType = requestSettings.requestHeaders["Content-Type"];
            }
            foreach (KeyValuePair<string, string> kvp in requestSettings.requestHeaders)
            {
                if (kvp.Key.Equals("Accept") || kvp.Key.Equals("Content-Type"))
                {
                    continue;
                }
                request.Headers.Add(kvp.Key, kvp.Value);
            }

            request.ContentLength = data.Length;

            // See https://msdn.microsoft.com/en-us/library/system.net.servicepoint.expect100continue(v=vs.110).aspx
            // A2 currently does not support this 100-Continue response for POST requests.
            request.ServicePoint.Expect100Continue = false;

#if ASYNC
            Stream stream = await request.GetRequestStreamAsync();
            await stream.WriteAsync(data, 0, data.Length);
            stream.Close();
#else
            Stream stream = request.GetRequestStream();
            stream.Write(data, 0, data.Length);
            stream.Close();
```

```
#endif
            }
            else
            {
               foreach (KeyValuePair<string, string> kvp in requestSettings.requestHeaders)
               {
                  if (kvp.Key.Equals("Accept") || kvp.Key.Equals("Content-Type"))
                  {
                     continue;
                  }
                  request.Headers.Add(kvp.Key, kvp.Value);
               }
            }

#if ASYNC
            WebResponse response = await request.GetResponseAsync();
#else
            WebResponse response = request.GetResponse();
#endif
            if (response.Headers.HasKeys())
            {
               foreach (string key in response.Headers.AllKeys)
               {
                  result.responseHeaders.Add(key, response.Headers.Get(key));
               }
            }

            result.responseCode = (int)(response as HttpWebResponse).StatusCode;

            using (StreamReader reader = new StreamReader(response.GetResponseStream()))
            {
#if ASYNC
               result.body = await reader.ReadToEndAsync();
```

```
#else
                result.body = reader.ReadToEnd();
#endif
            }
        }
        catch (Exception e)
        {
            result.responsMessage = e.Message;
            Log(Severity.Error, String.Format("{0} - {1}", e.GetType().Name, e.Message));
        }
        return result;
    }

#endregion IWebServiceRequest Members
```

ILog for Utily3D/PCL/Xamarin/.Net 4.5+

Note: The LogLevel in the if statement can be adjusted to raise or lower the severity being logged.

Note: The Debug.WriteLine statement might need adaption to the Game Engine and Platform.

```
#region ILog Members

    /// <summary>
    /// Executes the log operation.
    ///
    /// Implement this in Game Engine Code.
    /// </summary>
    ///
    /// <param name="severity"> The severity. </param>
    /// <param name="msg">      The message. </param>
    public void Log(Severity severity, string msg)
    {
       if (((int)LogLevel.Info & (int)severity) == (int)severity)
       {
          if (String.IsNullOrEmpty(msg))
          {
             Debug.WriteLine("");
          }
          else
          {
             Debug.WriteLine(String.Format("{0}: {1}", severity, msg));
          }
       }
    }


#endregion ILog Members
```