URBAN BRAIN STUDIOS

Chat Mapper Exporter SDK

Quick Start Guide

Version 2.0

2/5/2016

Contents

Welcome

Thank you for purchasing a commercial license of Chat Mapper. Included with your purchase is access to the Chat Mapper SDK, which allows you to develop your own exporters using .NET. For this guide, we will assume you are using Visual Studio 2008, but the steps are similar for other development environments and versions. Before getting started, make sure you have the required prerequisites:

- ✓ .NET Framework 3.5 SP1 installed
- ✓ Chat Mapper 1.x Installed with commercial license activated
- ✓ Development environment (preferably Visual Studio)
- ✓ ChatMapperAPI.DLL that came with the download

Installation

Installation of the SDK is a very simple process.

1) If you have not already done so, make sure your installation of Chat Mapper is licensed. You can check this by clicking **Help | Enter License Key** from the Chat Mapper program menu.

2) Copy or move the **ChatMapperAPI.DLL** file into your Chat Mapper installation directory. Typically this is **C:\Program Files\Chat Mapper**

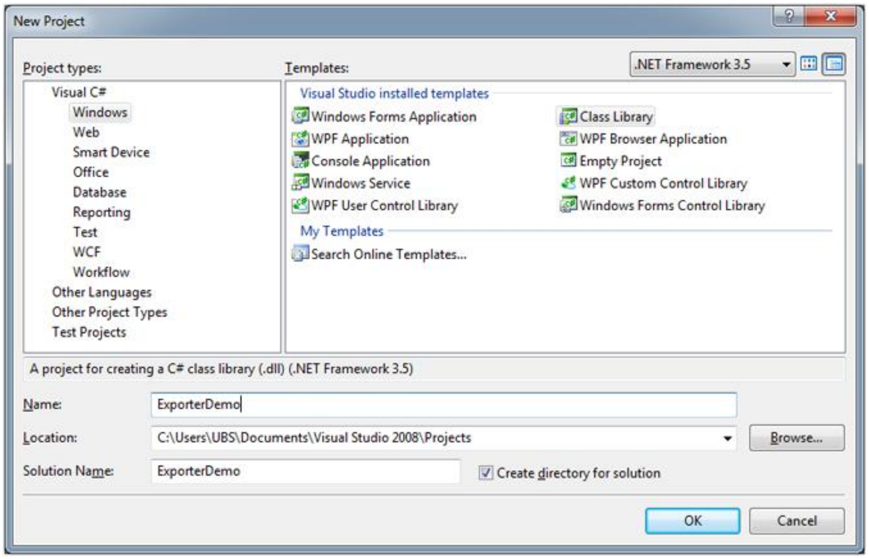
3) Ensure that the installation directory also includes **UrbanBrain.ChatMapper.Interfaces.DLL** and **System.ComponentModel.Composition.DLL** as well. These files will be added as references in the next section.

Starting a New Project

In this section, you will be up and running and ready to write the code for your exporter. We will:

- ✓ Start a new .NET class library project using .NET Framework 3.5
- ✓ Add necessary file references and using statements
- ✓ Implement the IChatMapperExporter interface
- ✓ Add an Export attribute to the exporter class

1) Open Visual Studio and start a new class library project with the .NET Framework 3.5 target selected. The language here does not matter, but C# examples will be shown from this point. Give your project a name and press OK. The project setup is shown in Figure 1.



.NET Framework 3.5

A project for creating a C# class library (.dll) (.NET Framework 3.5)

Name: ExporterDemo

Location: C:\Users\UBS\Documents\Visual Studio 2008\Projects

Solution Name: ExporterDemo

☑ Create directory for solution

Figure 1. New project setup for exporter.

The project is created with a default class file called Class1. Rename this class to match the name of your namespace. This is not required but makes things tidier.

2) **Right-click** the References section in the Solution Explorer and choose **Add Reference…** and browse to select the following three DLL files, all found in your Chat Mapper installation directory:

- ✓ ChatMapperAPI.dll
- ✓ System.ComponentModel.Composition.dll
- ✓ UrbanBrain.ChatMapper.Interfaces.dll

Also set the **Copy Local** property for these references to **false** so that your build folder is not cluttered with other DLL files. You can then add the following using statements to the top of your class file:

```
using UrbanBrain.ChatMapper.API;
using UrbanBrain.ChatMapper.Interfaces;
using System.ComponentModel.Composition;
```

The first line enables use of all of the API methods to access the live data in the opened Chat Mapper project, detailed in the **Chat Mapper API Documentation.chm** reference file. The second line allows access to all of the data structure interfaces that Chat Mapper uses. The third using enables use of the Managed Extensibility Framework (MEF) that will allow Chat Mapper to discover and use your compiled exporter DLL.

Note: In order to compile the examples later in this guide, you will also need to add the System.Windows using as we will be using the built-in MessageBox class. This requires the PresentationFramework reference from the .NET tab to be added to the project as well.

3) You are now ready to implement the **IChatMapperExporter** interface, which defines the required properties and methods of your exporter class. Add the **IChatMapperExporter** interface definition so that your class definition looks like this:

```
public class ChatMapperExporterDemo : IChatMapperExporter
```

Then, **Right-Click** the word **IChatMapperExporter** and select **Implement Interface | Implement Interface** from the popup context menu. This will cause all of the necessary properties and a single Run method to be added.

4) Finally, above the class definition, add an Export attribute in order to link your exporter to the main Chat Mapper application:

```
[Export("/App/Exporters", typeof(IChatMapperExporter))]
```

Here is what your file should look like at this point with the properties filled in:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UrbanBrain.ChatMapper.API;
using UrbanBrain.ChatMapper.Interfaces;
using System.ComponentModel.Composition;
using System.Windows;

namespace ChatMapperExporterDemo
{
    [Export("/App/Exporters", typeof(IChatMapperExporter))]
    public class ChatMapperExporterDemo : IChatMapperExporter
    {
        public string Author
        {
            get { return "Chatmapper Studios"; }
        }

        public string Description
        {
            get { return "Runs some demonstration code."; }
        }

        public string Title
        {
            get { return "Demo Exporter"; }
        }

        public string Version
        {
            get { return "1.0"; }
        }

        public bool Run(params object[] args)
        {
            throw new NotImplementedException();
        }
    }
}
```

Adding Your Exporter Code

The last step before compiling and testing your exporter is to implement the main code section, which will do the work for your exporter. When your exporter is selected from the export menu in Chat Mapper, the Run method will be called. Although the Run method accepts an array of arguments, at this time, none are passed from Chat Mapper. If the method returns true, it indicates a successful execution of the exporter. As such, we recommend always surrounding code in the Run method with a try-catch statement like this:

```
public bool Run(params object[] args)
{
    try
    {
        // Export code goes here
        return true;
    }
    catch { return false; }
}
```

Examples Using the API

At this point, you should glance through the API documentation contained within **Chat Mapper API Documentation.chm** file that was included with your download. You will primarily be interfacing with the Chat Mapper project data by using the methods in the static class **ChatMapperAPI**. The following examples show a small portion of the available API features.

Example 1 – Displaying a list of actors in the project

This example iterates through each of the project actor assets and prints their names.

```
string result = "";
foreach (IAsset a in ChatMapperAPI.GetAssetList(AssetTypes.Actor))
    result += a.GetAssetField((int)ActorFields.Name).GetValue().ToString();
result += "\n";
MessageBox.Show(result);
```

The **IAsset** interface is used to define all asset objects except for conversations and dialogue nodes. Conversations and dialogue nodes implement the **IConversation** and **IDialogNode** interfaces respectively. **ChatMapperAPI.GetAssetList** method returns a list of assets of the given type, which can then be looped through.

Each asset interface contains a **GetAssetField** method which can be used to access the asset field at the given index. Enums for each type of asset are provided for convenience for the required assets, but these must be cast as integers to specify the field index. After getting an asset field, the field value can be accessed using the **GetValue** method.

Example 2 – Displaying a list of selected node's audio files

This example gets the currently selected dialogue node from the conversation tree and displays a list of all audio files that have been added to the selected node.

```
string result = "";
IDialogNode node = ChatMapperAPI.GetSelectedDialogNode();
if (node != null)
{
    result += "Audio Files for ";
    result += node.GetAssetField((int)DialogFields.Title).GetValue().ToString();
    result += ":\n";
    List<string> audioFiles =
        (List<string>)node.GetAssetField((int)DialogFields.AudioFiles).GetValue();
    foreach (string file in audioFiles)
        result += file + Environment.NewLine;
}
MessageBox.Show(result);
```

Here we use the **GetSelectedDialogNode** method to retrieve the currently selected node in the conversation tree, which follows the **IDialogNode** interface. The title of the node is retrieved much like the actor names in Example 1.

Any asset field that has a type of **AssetFieldTypes.Files** must have the value cast to a **List<string>** as shown. After the cast, each string of the list can be looped through and represent the filenames defined in Chat Mapper.

Example 3 – Parsing Dialogue Markup Tags

In this example, we will split up a dialogue node text into segments as defined by any "break" tags, and then see the individual steps that are present in each segment.

```
string result = "";
IDialogNode node = ChatMapperAPI.GetSelectedDialogNode();
List<DialogSegment> dialogSegments = ChatMapperAPI.ParseDialogText(node);
    foreach (DialogSegment segment in dialogSegments)
    {
        foreach (DialogStep step in dialogSegments[0].Steps)
        {
            if (step.StepType == DialogStepType.Text)
                result += "Display Text: " + step.Data + "\n";
            else if (step.StepType == DialogStepType.Picture)
                result += "Display Picture " + step.Data + "\n";
        }
    }
MessageBox.Show(result);
```

Here we use the **ParseDialogText** method to get a list of segments, each of which contains a list of steps. Currently, the steps can either be text or picture steps. If the step is a text step, then the data property of the step contains the text to be displayed. Similarly, if the step is a picture step, then the data property contains the filename of the picture to display.

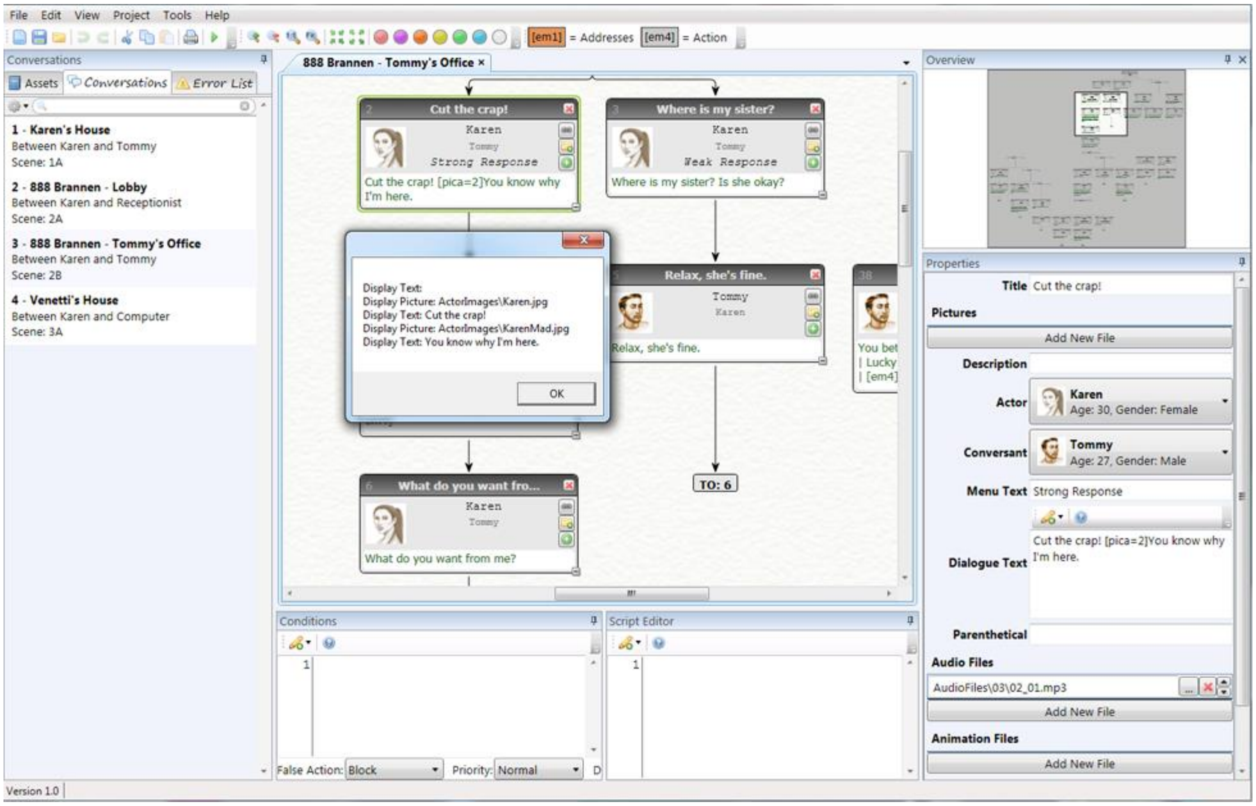
Compiling and Testing Your Exporter

After you complete your exporter, you are ready to compile and test.

1) Build the solution, which will create the exporter DLL in the default output directory set by Visual Studio.

2) Copy this file to the Exporters directory inside the Chat Mapper installation directory. This is usually **C:\Program Files\Chat Mapper\Exporters**

3) Run Chat Mapper and go to the **File | Export** menu option and you should see your exporter near the bottom of the list. Click it to run the exporter code. Figure 2 shows the results from Example 3.



```
Display Text:
Display Picture: ActorImages\Karen.jpg
Display Text: Cut the crap!
Display Picture: ActorImages\KarenMad.jpg
Display Text: You know why I'm here.
```

1 - Karen's House
Between Karen and Tommy
Scene: 1A

2 - 888 Brannen - Lobby
Between Karen and Receptionist
Scene: 2A

3 - 888 Brannen - Tommy's Office
Between Karen and Tommy
Scene: 2B

4 - Venetti's House
Between Karen and Computer
Scene: 3A

Title: Cut the crap!
Actor: Karen, Age: 30, Gender: Female
Conversant: Tommy, Age: 27, Gender: Male
Menu Text: Strong Response
Dialogue Text: Cut the crap! [pica=2]You know why I'm here.
Audio Files: AudioFiles\03\02_01.mp3
False Action: Block
Priority: Normal

Figure 2. Result of Example 3 exporter.

Note that in general, the default actor image is added on as the first segment.

Summary

We hope that these examples show just how easy it is to get started writing your own exporters. The API reference includes all available methods and interface definitions at your disposal. If you have any additional questions or concerns, as always you can post your question on the discussion forums under the API discussion, or contact us directly at support@chatmapper.com