nexusgamesoft™

Table Of Contents

1.Introduction....4
2.Installation Guide....4
3.Getting Started....4
3.1.Basic Example....4
3.2. Advanced Example....5
3.3. Building for your plattform....6
3.3.1. Building Win/Mac/Linux executable....6
3.3.2. Building for Webplayer and WebGL....6
3.3.3. Building for Android and IOS....7
3.Package Content....8
4.1 Program # and AIML....8
4.2. Jurassic....8
4.3 Use modified Program # with Jurassic....8
4.3.1. Write Java Script into AIML....8
4.3.2. AIML SET and GET element....9
4.3.3. Get and set global variables....10
4.3.3.2 . Get and set global variables in the AIML file....11
4.3.3.3. Basic Chat Syntax....12
4.5. Chatbot framework....14
4.5.1. Introduction to Chatbot framework....14
4.5.2. Chatbot.Core class....16
4.5.3. Chatbot.Trigger class....16
4.5.4. Chatbot.Motive class....16
4.5.5. Chatbot.NestedPlanner class....16
4.5.6. Helper functions....16
4.5.7. ChatbotCore MonoBehaviour....17
4.5.8. User Interfaces....17
4.5.9. Setting Component....17
4.5.10. Trigger Component....17

4.5.11. Motive Component.....17
4.5.12. Planner Component.....17
4.5.13. How Components have to be arranged.....17
1.Troubleshooting.....18
2.Feedback.....18

1. Introduction

Thank you very much for downloading this asset. Chatbot is a chatbot solution for Unity, based on a proven chatbot system called AIML 1.0.1, **A**rtificial **I**ntelligence **M**arkup **L**anguage. In this asset modificated Programm # is used to parse AIML files. For more Information about Program# or AIML read chapter Program # and AIML. To extend the functionality of AIML and to make better interaction between the AIML files and Unity C# or Java Scripts(Unity Script), an additional runtime Java Script parser Jurassic (see chapter Jurassic) is implemented. To mediate the potential from extended AIML and game programming, in this case Unity, several example scenes are included.

For more information, f.a.q., articles, contact and feedback visit http://www.chatbotunityasset.com/. Don't forget to post reviews in the Unity Asset Store. Thank you in advance and much fun creating new games and more.

2. Installation Guide

1. Make sure, that you selected the Plattform PC, Mac & Linux Standalone, Webplayer or WebGl (Unity 5.2.0 upwards).You can do this by clicking File->Build Settings-> Click your Plattform once ->Switch Plattform.

2.When you selected Plattform PC, Mac, Linux or WebGL go to Edit->Project Settings->Player->Optimiation->Api Compatibility Level and select .Net 2.0.

Only when you use the Jurassic source code:

3.Copy the gmcs.rsp (Pc,Mac,Linux) and smcs (Webplayer) file from the Chatbot folder in the Assets root folder. The file contains -unsafe to enable unsafe mode.

4. Save your Scene and Project and restart Unity to adopt the unsafe setting. Else you will get the unsafe exception.

5. Now Chatbot is installed correct and you can continue with the first steps below.

3. Getting Started

3.1. Basic Example

The basic example is intended to show the basic chat function of the AIML interpreter Program C#. Open Basic example.scene and you should be able to run the application and chat with the chatbot.

To reproduce Basic example.scene follow following steps:

1. Create a new scene.

2.Create a new gameobject and attatch the AIMLTestChat.cs skript in the Chatbot/Unity Implementation/Example Scripts folder.

3.Run the scene and you should be able to chat with your chatbot.

Additional for Webplayer or WebGL plattform:

1. Be sure to switch to the Webplayer/WebGl plattform.

2. You need to add the Webplayer Component to your gameobject (if not already attatched). You can find it in Components->Chatbot->Program Sharp->Webplayer Component.

3. When you compile your example be sure to insert following folders to your release folder:

Chatbot/Program #/config

Chatbot/Program #/aiml

4. Make sure that all your files have no starting characters. To check this, you need to open your files with a hex editor and look, if the files start with the core xml content or if there are any other characters before this. Remove the starting characters. Else you will get errors when importing the files.

5. On WebGl plattform be sure to select .Net 2.0 Api as described before. Also change settings under Edit->Project Settings->Player->Publishing Settings:

WebGL Memory Size: 512 Works for the provided sample.

Enable Exceptions: Explicity Thrown Exceptions Only

3.2. Advanced Example

This example introduces you to the Chatbot framework. Simply open Advanced Example scene and hit play. If you want to reproduce the example open a new scene and drag the ChatbotSimpleEmoticonChat.prefab in your Scene. It is also possible to take the ChatbotRaw.prefab and attach the needed game objects and components yourself. Chapter 4.5.13. describes, how the Components have to be arranged.

Additional for Webplayer plattform:

1. Be sure to switch to the Webplayer plattform.

2. You need to add the Webplayer Component to your root chatbot gameobject (if not already attatched). You can find it in Components->Chatbot->Program Sharp->Webplayer Component.

3. When you compile your example be sure to insert following folders to your release folder:

Chatbot/Program #/config

Chatbot/Program #/aiml

Chatbot/Unity Implementation/Emoticons

4. Make sure that all your files have no starting characters. To check this, you need to open your files with a hex editor and look, if the files start with the core xml content or if there are any other characters before this. Remove the starting characters. Else you will get errors when importing the files.

5. On WebGl plattform be sure to select .Net 2.0 Api as described before. Also change settings under Edit->Project Settings->Player->Publishing Settings:

WebGL Memory Size: 256 Works for the provided sample.

Make sure you ran the scene once in the editor if you made changes to the gifs in the “Chatbot/Unity Implementation/Emoticon/” folder. They may need to be saved as single frames in the “Chatbot/Resources/Chatbot/Unity Implementation/Emoticons/” folder again.

3.3. Building for your plattform

Take care that all your aiml and config files have an appropriate file encoding. They are tagged at the beginning of every xml document.

<?xml version="1.0" encoding="utf-8" ?>

The encoding “utf-8” seems to work on every system. If you use e.g. “iso 8859-15” you might run into exceptions on Windows, because Net Framework does not support all encodings.

3.3.1. Building Win/Mac/Linux executable

After every compilation you need to place the aiml, config (and unity implementation for advanced example) files in the Executable_Data folder.

For the basic example you need to add the following folders/files:

- Executable_Data/Chatbot/Program #/aiml
- Executable_Data/Chatbot/Program #/config

For the advanced example you need the following folders/files:

- Executable_Data/Chatbot/Program #/advancedaiml
- Executable_Data/Chatbot/Program #/config

If you build the advanced example make sure you ran the scene once in the editor if you made changes to the gifs in the “Chatbot/Unity Implementation/Emoticon/” folder. They may need to be saved as single frames in the “Chatbot/Resources/Chatbot/Unity Implementation/Emoticons/” folder again.

For the advanced example make sure, that you selected Net 2.0 framework. Else your system.drawing.dll will lead to errors.

3.3.2. Building for Webplayer and WebGL

After compiling your executive you need to place the aiml and config files in the same folder as the html and unityplugin file.

For the basic example you need to add the following folders:

- Chatbot/Program #/aiml
- Chatbot/Program #/config

For the advanced example you need the following folders:

- Chatbot/Program #/advancedaiml
- Chatbot/Program #/config

If you build the advanced example make sure you ran the scene once in the editor under Windows/Linux/Mac plattform if you made changes to the gifs in the “Chatbot/Unity Implementation/Emoticon/” folder. They may need to be saved as single frames in the “Chatbot/Resources/Chatbot/Unity Implementation/Emoticons/” folder again.

Also make sure, that your config file contains all used .aiml files. Also check your .aiml and .config files to not have any file sign chars at the beginning of the files. If you edit and save your files, the editor might add those chars leading to xml read errors. To remove them use a hex file editor tool to remove those chars.

3.3.3. Building for Android and IOS

As the file I/O on Android is restricted it is a good solution to store our needed files in the Resources folder.

To do this, copy the following files:

For the basic example:

- Chatbot/Program #/aiml => Chatbot/Resources/Chatbot/Program #/aiml
- Chatbot/Program #/config => Chatbot/Resources/Chatbot/Program #/config

For the advanced example:

- Chatbot/Program #/advancedaiml => Chatbot/Resources/Chatbot/Program #/advancedaiml
- Chatbot/Program #/config => Chatbot/Resources/Chatbot/Program #/config
- Emoticons folder is created and filled when entering playmode in the Unity Editor

Unity does not recognice aiml files as text files. Thus you need to **resave all .aiml files to .xml files**.

If you build the advanced example make sure you ran the scene once in the editor if you made changes to the gifs in the “Chatbot/Unity Implementation/Emoticon/” folder. They may need to be saved as single frames in the “Chatbot/Resources/Chatbot/Unity Implementation/Emoticons/” folder again.

3. Package Content

4.1 Program # and AIML

AIML is as a XML dialect and was developed by Richard Wallace and a worldwide free software community between 1995 and 2002. AIML formed the basis for what was initially a highly extended Eliza called "A.L.I.C.E." ("Artificial Linguistic Internet Computer Entity"), which won the annual Loebner Prize Competition in Artificial Intelligence three times, and was also the Chatterbox Challenge Champion in 2004.

Because the A.L.I.C.E. AIML set was released under the GNU GPL, and because most AIML interpreters are offered under a free or open source license, many "Alicebot clones" have been created based upon the original implementation of the program and its AIML knowledge base. Free AIML sets in several languages have been developed and made available by the user community. There are AIML interpreters available in Java, Ruby, Python, C++, C#, Pascal, and other languages. A semi-formal specification and a W3C XML Schema for AIML are available.

For more Information see http://www.alicebot.org/

For more Information about the unmodified Program # see http://aimlbot.sourceforge.net/

4.2. Jurassic

Jurassic is an implementation of the ECMAScript language and runtime. It aims to provide the best performing and most standards-compliant implementation of JavaScript for .NET. Jurassic is not intended for end-users; instead it is intended to be integrated into .NET programs. If you are the author of a .NET program, you can use Jurassic to compile and execute JavaScript code.

Features:

- Supports all ECMAScript 3 and ECMAScript 5 functionality, including ES5 strict mode
- Well tested - passes over five thousand unit tests (with over thirty thousand asserts)
- Simple yet powerful API
- Compiles JavaScript into .NET bytecode (CIL); not an interpreter
- Deployed as a single .NET assembly (no native code)
- Basic support for integrated debugging within Visual Studio
- Uses light-weight code generation, so generated code is fully garbage collected
- Tested on .NET 3.5, .NET 4 and Silverlight

4.3 Use modified Program # with Jurassic

4.3.1. Write Java Script into AIML

The structure of an valid AIML document could be as described below.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<aiml>
<category>
        <pattern>WHY </pattern>
        <template>Because</template>
</category>
</aiml>
```

For more Information about the AIML 1.0.1 draft see http://www.alicebot.org/TR/2005/WD-aiml/WD-aiml-1.0.1-008.html#section-terminology

To implement a Java Script write the following:

```
        <?xml version="1.0" encoding="ISO-8859-15"?>
        <aiml>
        <category>
                <pattern>WHY </pattern>
                <template>Because
                <script>
                function main(){
                        return “”;
                }
                </script>
                </template>
        </category>
        </aiml>
```

The Script element is treated like any other aiml element and returns a Value to the template element. Pay attention, when writing return; because this returns 0 instead of “”.

4.3.2. AIML SET and GET element

Usually the SET and GET element gets and sets variables in the user predicates.

```
        // Old set.cs code:
```

```
this.user.Predicates.addSetting(this.templateNode.Attributes[0].Value,
this.templateNode.InnerText);

return this.user.Predicates.grabSetting(this.templateNode.Attributes[0].Value);
```

To handle and synchronise all Variables in the bot and Jurassic global variables, bot global variables are used instead of player predicates:

```
// New set.cs code:

// Set Global Variable in Jurassic

this.bot.jscript_engine.SetGlobalValue(this.templateNode.Attributes[0].Value,this.templateN
ode.InnerText);

// Set Global Variable in Program #

if(this.bot.GlobalSettings.containsSettingCalled(this.templateNode.Attributes[0].Value))

this.bot.GlobalSettings.removeSetting (this.templateNode.Attributes[0].Value);

this.bot.GlobalSettings.addSetting (this.templateNode.Attributes[0].Value,
this.templateNode.InnerText);
```

```
return this.bot.GlobalSettings.grabSetting(this.templateNode.Attributes[0].Value);
```

Now you are able to set and get the same global variables in AIML and Jurassic.

4.3.3. Get and set global variables

Keep in mind, that you need to reach Program # and Jurassic global variables when setting or getting global variables.

4.3.3.1. Get and set global variables within Unity

To set global variables in Unity use following code:

```
String name,value=”Your name and value”;

// Set Global Variable in Jurassic

jscript_engine.SetGlobalValue(name,value);

// Set Global Variable in Program #

if(this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.containsSettingCalled(name))

this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.removeSetting (name);

this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.addSetting (name, value);
```

To get global variables in Unity use:

```
// Get Global Value from Jurassic
```

```
value =
this.AIMLAdvancedTestChatInstance.bot.jscript_engine.GetGlobalValue<string>(name);

// Set Global Variable in Program # to accomplish same global variables in both Jurassic
and // modified Program #

if(this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.containsSettingCalled(name))

        this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.removeSetting (name);

this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.addSetting (name, value);
```

4.3.3.2 . Get and set global variables in the AIML file

In the core AIML language you can use <set name="YourVariable">Value</set> to set a global variable in the Program # bot global values and the Jurassic global variables at the same time.

Use <get name="YourVariable">Value</get> to get a global value from the bot global variables.

In the Javascript use following:

```
function main(){

        // To get a global variable just use it's name

        var newvalue = yourvariable;

        // or by string

        newvalue=this["yourvariable"];

        return"";

}
```

To set an existing global Variable in Script:

```
function main(){

        //Set an existing global variable

         Yourvariable="YourValue";

        return"";

}
```

You need to grab the global Jurassic settings after processing chat() and pass them to the modified Program # like in the AIMLTestChat.cs file:

```
// Prepare Variables

//bot.jscript_engine.SetGlobalValue("abc",15);
```

```
request.rawInput = Input_Text;

request.StartedOn = DateTime.Now;

result = bot.Chat(request);

Output_Text = result.Output;

Input_Text = "";


// Gather Variables

String tmpabc = bot.jscript_engine.GetGlobalValue<string>("abc");

// Set Global Variable in Program #

if(this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.containsSettingCalled(“abc”))

this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.removeSetting (“abc”);

this.AIMLAdvancedTestChatInstance.bot.GlobalSettings.addSetting (“abc”, tmpabc);
```

4.3.3.3. Basic Chat Syntax

The following AIMLTestChat.cs script is an example of how to use Program # and Jurassic to chat with the bot:

```
using System;

using UnityEngine;

using System.Collections;

using AIMLbot.Utils;


public class AIMLTestChat : MonoBehaviour {

        private string Input_Text = "";

        private string Output_Text ="";

        private AIMLbot.Bot bot;

        private AIMLbot.User user;

        private AIMLbot.Request request;

        private AIMLbot.Result result;
```

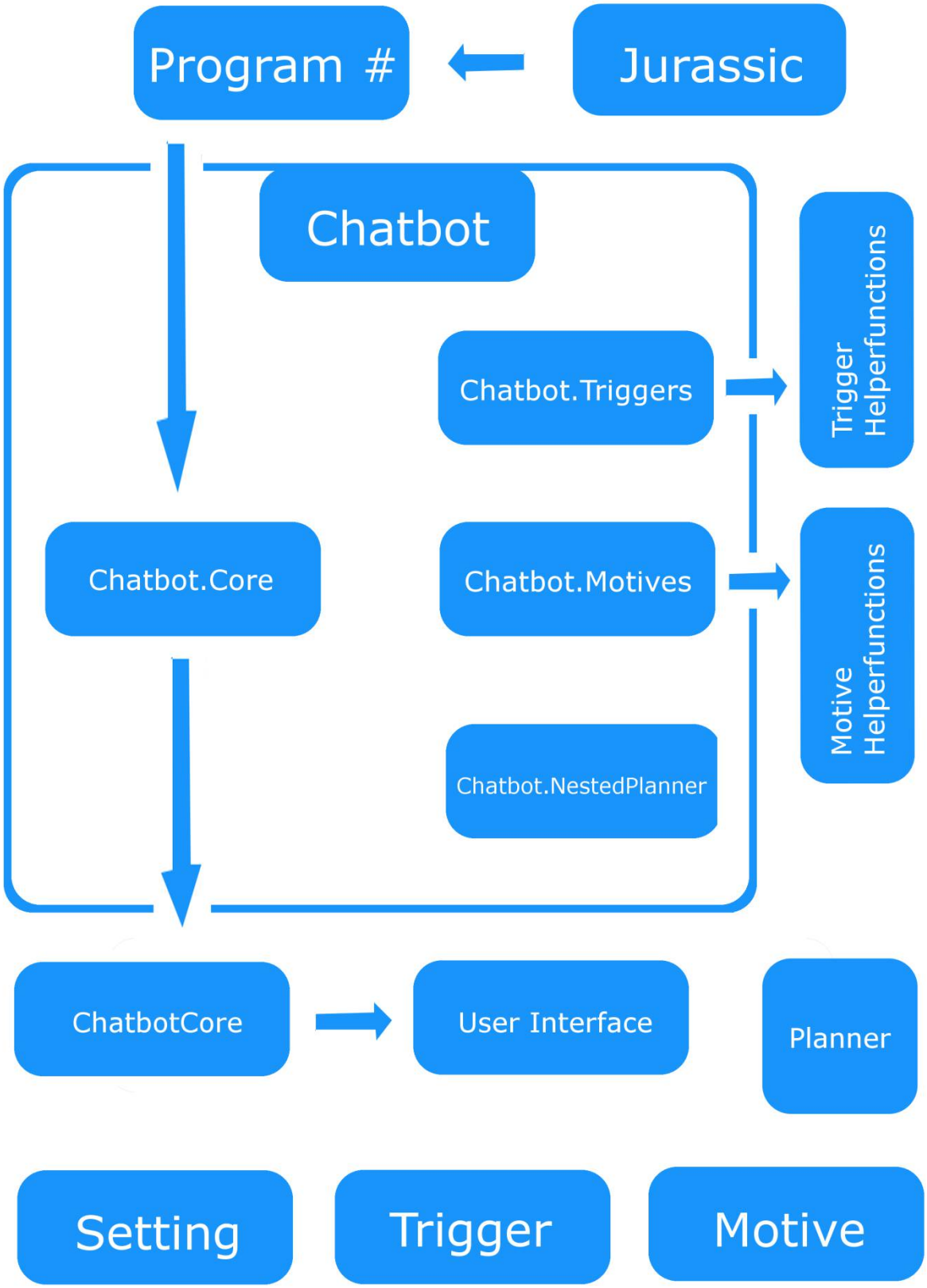
```
// Use this for initialization
void Start () {
        bot = new AIMLbot.Bot();
        user = new AIMLbot.User("MyUser",bot);
        request = new AIMLbot.Request("",user,bot);
        result = new AIMLbot.Result(user,bot,request);



        bot.loadSettings(Application.dataPath + "/Chatbot/config/Settings.xml");
        // Load AIML files
        bot.loadAIMLFromFiles();

}

void OnGUI () {
        // Enable Word warp
        GUI.skin.label.wordWrap = true;
        // Make a background box
        GUI.Box(new Rect(10,10,300,150), "Chat with a Chatbot");
        // Make output label
        GUI.Label (new Rect (20, 30, 280, 40), Output_Text);
        // Make a text field that modifies Input_Text.
        Input_Text = GUI.TextField (new Rect (20, 100, 280, 20), Input_Text, 100);

        // If send button or enter pressed
        if(((Event.current.keyCode == KeyCode.Return)||GUI.Button(new
        Rect(250,130,50,20),"Send")) && (Input_Text != "")) {
```

```
                        // Prepare Variables
                        //bot.jscript_engine.SetGlobalValue("abc",15);

                        request.rawInput = Input_Text;
                        request.StartedOn = DateTime.Now;
                        result = bot.Chat(request);
                        Output_Text = result.Output;
                        Input_Text = "";

                        // Gather Variables
                        // bot.jscript_engine.GetGlobalValue<string>("abc");
                    }
                }

                // Update is called once per frame
                void Update () {

                }
            }
```

4.5. Chatbot framework

4.5.1. Introduction to Chatbot framework

As the aim of this asset is to provide natural language programming to Unity and thus make Program # together with Jurassic more useful for game programming, Chatbot framework is the solution to simplify and enlarge the capabilities of this asset. Let's start with an overview graphic.



We already discussed Program # and Jurassic in chapter Program # and AIML-Use modified Program # with Jurassic. These basic libraries are not affected by Chatbot framework and thus can still be used

to create your AIML 1.0.1 chat application as shown in Basic Example scene. The Advanced Example Scene introduces you to the Chatbot framework, that is based upon Program # and Jurassic.

4.5.2. Chatbot.Core class

This class comprises all Chatbot information and grants easy handling of initialization, global settings, chatting and more. It is instantiated in the ChatbotCore MonoBehavior. See more in chapter …

4.5.3. Chatbot.Trigger class

Comprise all Trigger information and processing algorithms. Retrieves trigger settings from Trigger MonoBehaviour. See more in chapter ..

4.5.4. Chatbot.Motive class

Contains all Motive information and processing functions. Retrieves motive settings from Setting MonoBehaviour. More information here.

4.5.5. Chatbot.NestedPlanner class

This class handles the ordered execution of motive functions. It's theoretically based upon utility theory. Motives can be nested. At first all motives to consider are selected. In case one, when last motive has the Boolean setting TriggerChildrenBeforeContinue set to false, all motives that have no child transforms are selected after last motive was executed. In case two, when last motive has the Boolean setting TriggerChildrenBeforeContinue set to true, all motives that have no child transforms and are children of last motive are selected after last motive was executed. Now the score of every selected motive, between 0 and 1, is calculated and the motive with the biggest score is selected to be the next motive to trigger. The formula for the score is Importance*Need*RelativExpectedTimeSpan, whereat every value is between 0 and 1.

4.5.5.1. Importance

Importance is a factor between 0 and 1, whereat 0 is equal to not important and 1 is equal to important.

4.5.5.2. Need

The individual need to execute this motive, whereat 0 is equal to no need and 1 is equal to need.

4.5.5.3. RelativeExpectedTimeSpan

This value represents the relative time span, the motive will require referred to all selected motives. Its value is between 0 and 1. It is calculated by dividing the motives expected time span in seconds by the sum of all selected motives time spans in seconds.

4.5.6. Helper functions

Helper functions aim at implementing Chatbot in the developers environment. Therefore they enable the developer to write their own Motive or Trigger helper scripts and connect them to Chatbot by using SendMessage function. E.g. you can connect PlayMaker to Chatbot framework.

4.5.6.1. Trigger helper functions

The trigger helper functions are individual scripts and execute upon individual needs. E.g. the TriggerPlayOnline script always executes, when the user moves the mouse, clicks or uses the keyboard. Or the TriggerFacialExpression script executes, if all entered global settings contain "1" and other global settings don't contain "1". When executed, the trigger helper function calls the

TriggerFromHelperFunction, which is member of the ChatbotCore component. With Playmaker you could e.g. send a message to the game object containing the component ChatbotCore:

SendMessage("TriggerFromHelperFunction","Name of the target trigger");

4.5.6.2. Motive helper functions

This function works similar to the trigger helper function. It is able to receive and send messages to the ChatbotCore Component. It aims at executing the motive individual with developer's scripts or e.g. Playmaker. It has a function called Trigger, which is called by Chatbot framework. The MotiveFacialExpression MonoBehaviour for example changes emoticon path, waits several seconds and then changes path back to default. When finished processing, the function **must** inform the ChatbotCore component, that it has finished executing, else the Chatbot framework waits in vain. It does this by calling the MotiveFinished function within ChatbotCore component. You can again do this via SendMessage("MotiveFinished").

4.5.7. ChatbotCore MonoBehaviour

This component is designated to be attatched to the Chatbot root game object. It is the interface between Chatbot framework and Unity Scene and MonoBehaviour script components.

4.5.8. User Interfaces

User interface components have to be attached to the same game object as the ChatbotCore component. Their function is to provide a graphical interface, where the user can input text and retrieve chat answers. It is intended to be easily modified by the developer. The user interface component used in the Advanced Example scene is called SimpleEmoticonChat.

4.5.9. Setting Component

The Setting Component has to be attached to any child game object of the game object containing ChatbotCore component. When adding a Setting component, you add a global chatbot setting at once. It is synchronized with the Chatbot.Core instance.

4.5.10. Trigger Component

The trigger component is synchronized with the Chatbot.Triggers instance and thus officiates as simple interface for the developer.

4.5.11. Motive Component

Must be attached to a game object, which is child of game object with AssignedMotives component attached. This is to separate the Motive templates from the motives handled in planner child game objects. It is again an interface between Unity scene and Chatbot framework.

4.5.12. Planner Component

Must be attached to a game object, which is child of game object with ChatbotCore component attached. This is an anchor for Chatbot framework to know where to handle planner motives.

4.5.13. How Components have to be arranged

Look at the Advanced Example prefab ChatbotSimpleEmoticonChat.prefab to see, how the gameobjects and components should be arranged.

4.5.13.1. Main Components

The main chatbot components are the ChatbotCore.cs script and one User interface script, e.g. SimpleEmoticonChat.cs. Those two need to be attatched to the main gameobject of your bot.

4.5.13.2. Sub Components

Mind Control game object is an optional game object, to keep it well-arranged. Also helper functions in generally can be located everywhere. The provided helper functions however need to be child game objects of the game object with ChatbotCore component attatched. The child game objects of Mind Control can be devided in global settings, trigger functions, motive templates and planner. The game object holding the motive templates must have a AssignedMotives components to let Chatbot framework know, where template motives are and where the motives attached to the planner game object are. Thus, you also need to assign the planner component to the game object that is to be holding motives to work off. Furthermore it makes sense to attach game objects with Setting component to a game object, that keeps all setting game objects.

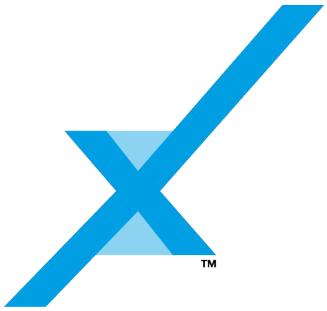
1. Troubleshooting

If you have any questions or issues, write an email at mail@chatbotasset.com or visit the forum www.chatbotunityasset.com/forum/.

2. Feedback

Please don't forget to give feedback and consider, that the project is still in continous development.

Thank you in Advance!



Nexus Gamesoft