Media Semantics Character Server
User's Guide

Contents

Getting Started **4**
Overview 4
Requirements 4
Installation 4
Where to start 4
Obtaining technical support 4

Character Server **5**
Using the CGI interface 5
Using the COM interface 6
WScript 7
ASP 7
ASP.NET 7
PHP 7
C++ 8
C# 8
CharacterServer.ini 8
extendedlog 9
ignorecached 9
maxflashcache 9
maximagecache 9
librarypath 9
projectpath 9
maxvideoframes 10
maxthreads 10
decodeflv 10
mp4, wmv, mov, flv, 3gp, jpg 10

Speech Server **11**
Overview 11
SpeechServer.ini 11
extendedlog 11
ignorecached 11
maxwavecache 11
maxchars 11
audioquality 11
channelN 12
simulateportlimitations 12
loquendolexicon 12
neolexicon 12
map 12
Configuring Channels 12

Face Server **14**
Overview 14
FaceServer.ini 14
extendedlog 14
avgsecs 14
failsafe 14

Program AB Adapter **15**

Overview 15
ProgramABAdapter.ini 15
extendedlog 15
maxthreads 15
javapath 15
programabpath 15
botspath 15
ailogstore 15
ailogserver 15
ailoguser 16
ailogpassword 16
sessionnotifyurl 16
sessionnotifyminrequests 16
Conversation management using a database 16
Sessions Table 16
Variables Table 16
Responses Table 16
Generating session emails 17
Operation & Performance 19
Maintaining the server 19
The stress tool 19
Speech licensing and throughput 19
Character licensing and throughput 20
Securing the server 20
Appendix A – COM API 22
Creation 22
Supporting methods 22
Rendering 23
Async render 24
Audio processing 25
Face design 26
Photofit 28
Photomorph 29

Getting Started

Overview

The Character Builder is an Integrated Development Environment, or IDE, for creating character-driven applications. You can specify content in a visual manner using the Builder's Outline, Design, and Script Views. These are merely graphical views on a rich underlying markup language, which can also be edited directly in the optional XML View. Markup tags are based on existing standards where possible, but several additional tags are introduced to specify character behavior. The underlying XML tag set forms a high-level application model. The Character Builder lets you create non-linear media, where the delivery is adapted to and influenced by the user. The Character Server takes this a step further by using server-side Text to Speech to create character streams "on the fly", using text entered by the user or derived from a database or web service.

Requirements

The Character Server requires Windows Server (2003 or higher) or Windows XP, Vista, or 7, or 8.

Installation

To install the Character Server, please run the associated installer program. The Server will install by default under "C:\Program Files (x86)\Character Server".

IMPORTANT: Be sure to read the "Readme" file, as it contains important post-installation, self-test, and licensing information.

Where to start

The Character Server is designed to complement the Character Builder. After the Character Server has been installed correctly, as verified by the proper operation of the self-test in the Readme, you will be ready to use the Character Builder New > Project wizard templates that require the Character Server.

Obtaining technical support

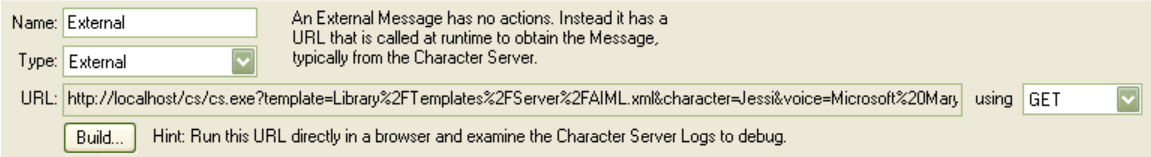
Registered users of the Character Server may access technical support by email at support@mediasemantics.com. Before contacting technical support, please be sure to visit our support area, at http://www.mediasemantics.com/support.htm. Here you will also find a link to our support board.

Character Server

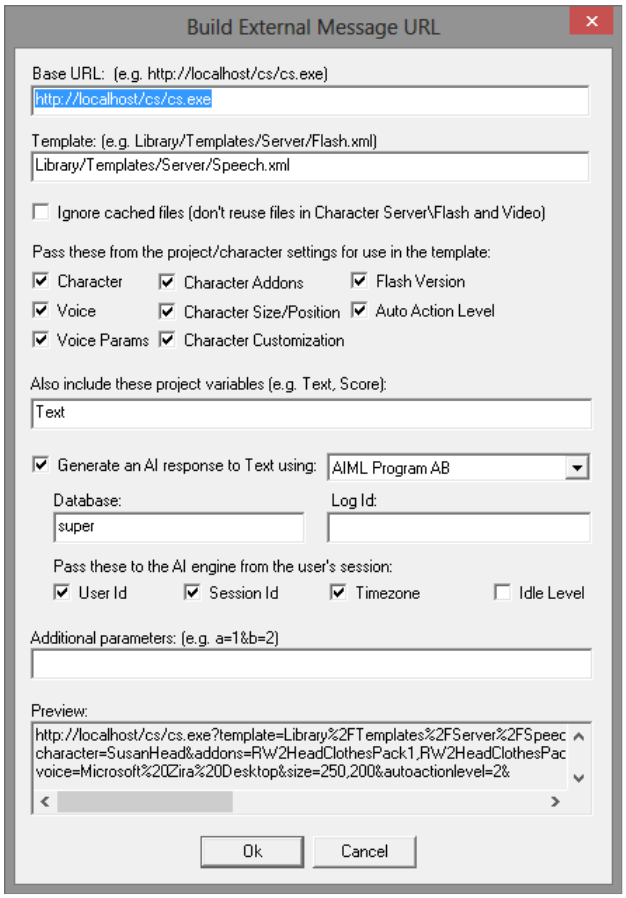
Using the CGI interface

The CGI (Component Gateway Interface) API is normally used by Character Builder-generated content in the form of a URL for an External Message. The External Message mechanism is particularly powerful, since it lets you combine static and dynamic content, and seamlessly transition from one to the other. For example a Character Builder scene might contain a background image, an initial image of the character, and an "idle" routine that is played over and over again to give the character the illusion of life. These might be combined with "just-in-time" generated Flash streams from a Character Server, using parameters known only at runtime.

The CGI API is accessed through the Character Builder's External Message Build URL dialog. Here is the Message panel in the Builder:



Pressing Build results in the following dialog:



All generation begins with a template file, located at the server. The rest of the options in this dialog let you specify typical values that will be used to instantiate the template. Many projects created by the New Project wizard will use the Speech.xml template. This is a very general template that lets you specify a character, voice, and message.

You will typically want to pass the name of the character and any customization parameters, so that the Server-generated character will look identical to the Builder-generated character – the checkboxes at the top of the dialog let you do this.

Your project will typically contain one or more variables, such as Text, whose contents are also passed to the server.

If you choose to, you can also specify an AI engine that will first interpret the text – the resulting response is then substituted in the template instead.

Finally, the Preview window lets you see what you url will look like.

The template file for an external message is a regular (XML) project file except that the loader attribute is set to 'external'. The template should have a single character and a single message. The file can include '$' placeholders such as '$text' and '$character'. These placeholders are filled in using values passed in the external file url.

After substitution, the xml is processed by the Server, and the resulting Flash stream is returned. You can invoke a cs.exe URL directly to verify the operation of the Server. If you have extended logging turned on, then your Character Server\Logs directory will contain an Extended Logs file that shows the actual instantiated XML that was processed, along with detailed timing and queue information.

Because of limitations in the cross-domain security model in Flash across browsers, content generated by the CGI interface must be consumed by a CB-generated file operating in the same domain. The CB-generated Flash file will ignore the domain portion of a cs.exe URL, and substitute its own domain instead.

For example, if a CB-generated Flash file 'flash.swf' contains an external message with the url *http://localhost/cs/cs.exe*, and this Flash file is further run from the url *http://www.acme.com/flash.swf*, then the actual URL used for the external message will be *http://www.acme.com/cs/cs.exe*, so as to enforce the same-domain rule. While the CB-generated Flash file must be located on the same domain as the Character Server, it can be referenced in an OBJECT tag in an HTML file located on a different domain.

Flash 8 content further distinguishes between flash files that are run in the file system "sandbox", e.g. 'c:\flash\flash.swf', and flash files in the network "sandbox", e.g. *http://www.acme.com/flash/flash.swf*. Since there is normally no overlap allowed between these two sandboxes, and since the CGI interface is inherently in the network sandbox, it follows then that CB-generated content with external messages should also be run from a network url, such as *http://localhost*, even during testing.

Using the COM interface

The CharacterServer.dll component exposes a COM API that can be accessed from many languages, including C/C++, .NET, ASP, and PHP. The COM API lets you access the same functionality as the cs.exe API, but with more control. The COM API also exposes additional functionality that is not available through the CGI API.

A complete description of the COM API is provided in Appendix A.

In order to use the COM API you will need a full Character Server license. Please contact sales@mediasemantics.com if you require a timed trial for the Character Server COM API.

WScript

You can test the COM interface outside of a web server by copying the following lines of code, pasting them into Notepad, and saving the result to a 'test.js' file. Place the file on your desktop and double-click it to run.

```
// COM interface example equivalent to readme self-test
server = WScript.CreateObject("CharacterServer.CharacterServer");
server.SetParameter("character", "TashaHead");
server.SetParameter("customhair", "tashahair");
server.SetParameter("customtop", "shirt1");
server.SetParameter("text", "Hello World!");
file = server.RenderToTempFile("Library/Templates/Server/Speech.xml");
WScript.Echo("Generated "+file);
```

When run, it should return a message such as "Generated CS123.swf".

The full COM API is described in the API Reference help file, available from the Character Builder Help menu.

ASP

Here is a sample ASP page that calls the Character Server's COM API.

```
<%@ Language=JScript %>
<%
Response.Expires = -1;  // Don't cache this
server = Server.CreateObject("CharacterServer.CharacterServer");
xml = "<project><stage><scene><text>Hello World</text></scene></stage></project>";
server.RenderAsResponse(xml); // Renders directly to the response stream
%>
```

ASP.NET

When running with ASP.NET, you should ensure that the ASP.NET Machine Account ("ASPNET" user) and the Launch IIS Process Account (IWAM_machinename), have write access to the Logs directory, since these accounts may be used to execute the COM component.

The COM API is a “classic COM interface”, so to call it from an ASP.NET page you should begin your page with <%@ Page aspcompat=true >. You can then create the Character Server object using the CreateObject command. For example:

```
<%@ Page aspcompat=true Language=VBScript ValidateRequest="false" %>
<%
Response.Expires = -1  ' Don't cache this
dim server, xml
server = Server.CreateObject("CharacterServer.CharacterServer")
xml = "<project><stage><text>Hello World</text></stage></project>"
server.RenderAsResponse(xml) ' Render directly to the response stream
%>
```

Note that the ValidateRequest option was also set to ‘false’ - this may be necessary if you are using an .aspx file as the External Message url for a Character Builder message with the POST option.

PHP

You can easily use PHP to validate and/or add to the set of GET parameters and then redirect to the CGI interface using the PHP header command:

```
header('Location: http://www.yoursite.com/cs/cgi.exe?parameters');
```

Alternatively you can call the COM interface:

```
<?php
$server = new COM("CharacterServer.CharacterServer") or die("Unable to load Server");
$xml = "<project><stage><text>Hello World</text></stage></project>";
$file = $server->RenderToTempFile($xml);
// replace this with a url to your flash directory
header('Location: http://www.yoursite.com/flash/' . $file);
?>
```

RenderAsResponse is only available under IIS - under Apache/PHP you can alias the Character Server's Flash (cache) directory as a virtual directory, then redirect to the resulting file.

C++

The easiest way to get started in Microsoft Visual C++ (with MFC support) is to use the Project > Add Class menu command, then under Visual C++/MFC, pick the “MFC Class From Typelib” tool. Then in the resulting dialog pick the type library “Media Semantics Character Server Type Library”, then select the interface ICharacterServer. The Class Wizard will create a CCharacterServer.h file with a CCharacterServer wrapper class. Here is a sample C++ function that uses this wrapper class:

```
#include "stdafx.h"
#include "CCharacterServer.h"
const CLSID CLSID_CharacterServer =
{0x3F8ECA7C,0xB371,0x4dad,{0x8B,0xA3,0x83,0x0B,0x93,0x17,0xC4,0xF9}};
void TestServer()
{
	CCharacterServer * pserver = new CCharacterServer();
	pserver->CreateDispatch(CLSID_CharacterServer);
	CString xml = "<project><stage><text>Hello World</text></stage></project>";
	CString file = pserver->RenderToTempFile(xml);
	// consume file
	delete pserver;
}
```

C#

In a .NET C# project, you can right-click on the Visual C# Solution Explorer's References folder and select Add Reference. On the COM tab, select “Media Semantics Character Server Type Library”. You can then use the following code:

```
using CharacterServerLib;

void TestServer()
{
	ICharacterServer server = new CharacterServerLib.CharacterServerClass();
	string xml = "<project><stage><text>Hello World</text></stage></project>";
	string file = server.RenderToTempFile(xml);
	// consume file
}
```

CharacterServer.ini

While the installer will have configured your system appropriately to begin development, there are a few parameters that can be tuned further. These parameters are specified in the CharacterServer.ini file, located in the product directory. This file is read upon startup of the service. **If you change any values, please be sure to stop and then start the service again for the changes to take effect.**

Here are the default contents of the CharacterServer.ini file:

```
[CharacterServer]
extendedlog=true
projectpath=c:\Documents and Settings\<username>\My Documents\Character Builder Projects
```

The possible entries are as follows:

extendedlog

When set to ‘true’, the Server maintains a log of all requests in the Character Server\Logs directory.

IMPORTANT: This setting is set to ‘true’ by the installer. You should set this to ‘false’ when the solution is running smoothly, or else the extended log file will become very large.

Defaults to ‘false’ if not specified.

ignorecached

When set to ‘true’, incoming requests are always rerendered, even if a request with the same xml is present in the cache. If you are using cs.exe (the only API under Character Server Standard), the outgoing response is also set to expire immediately, so a subsequent request does not go to the cache. This setting *can* be handy during initial installation and testing, but it is not recommended for general use. Defaults to ‘false’.

maxflashcache

Specifies the number of files to maintain in the Flash subdirectory as a cache. A render request with an identical XML string to a request in the cache will be satisfied immediately from the cache, providing ‘ignorecached’ is not set (see above). Defaults to 1000.

maximagecache

Specifies the number of files to maintain in the Images subdirectory as a cache. PhotoFit, PhotoMorph, and realistic characters that support clothing and hair options will cause the Server to create files in the image cache. When such a character is used for the first time, the Character Server will be slower, as it populates the image cache. Subsequent generations with that character will approach the maximum speed due to the availability of cached image files.

librarypath

Specifies the path to the library directory. By default, references of the form ‘Library/xyz’ will map to ‘C:\Program Files\Character Builder\Library\xyz’. You can change this when deploying to your production web server. For example, you might change this to ‘C:\Program Files\Character Server\Library’, and then upload to this location those portions of your Character Builder library that your project depends on.

If ‘librarypath’ is not specified, the Character Server will use the Library of the Character Builder, if installed, so you do NOT need to set this parameter in the recommended configuration, in which both the Builder and the Server are installed on the same server machine.

Note that character resources present in ‘C:\Documents and Settings\<user>\Application Data\Media Semantics\Library’ directory will take precedence over those in the regular library directory. This lets you install the Character Builder on the server and use the Help > Download Manager tool to download character resources.

projectpath

Specifies the path to the “projects” directory. References of the form Projects/xyz will be mapped using this setting in the same way that references beginning with ‘Library’ will be mapped using the ‘librarypath’ setting. The default value for ‘projectpath’ is set by the installer, and is designed to facilitate experimentation on a development machine. When deploying to a production web server, you might change this to the document root of the web site, such as ‘c:\inetpub\wwwroot’.

For security reasons, the CGI interface never allows absolute paths to files on the server to be specified in the CGI parameters – all references to server files and paths should begin with ‘Library’, ‘Projects’, or ‘Data’.

maxvideoframes

Specifies the maximum number of frames that will be rendered to video. Use 0 to never truncate video renders. This is an artificial limit designed to prevent misuse.

maxthreads

Specifies the maximum number of threads that will be created in order to service simultaneous requests. The default is 8, and should be adequate.

decodeflv

Specifies an optional command line used for the DecodeFLV API. Defaults to "[Character Server Folder]\ffmpeg.exe" -i $infile $outfile

mp4, wmv, mov, flv, 3gp, jpg

Specifies optional command lines used for the RenderVideo API. Defaults to "[Character Server Folder]\ffmpeg.exe" -i $infile -r $framerate $params $outfile

Speech Server

Overview

The Speech Server is a Windows service that is called by the Character Server to perform Text-to-Speech and speech compression operations. Both the Speech Server and the Character Server must be running in order to service most requests. The Speech Server keeps instances of one or more text-to-speech engines open and queues up text-to-speech requests. Requests are often filled instantly through the use of a cache of .wav files located in the 'Wave' subdirectory.

The Speech Server is designed to maximize the use any Text to Speech speech licenses that you may have installed in the system.

SpeechServer.ini

The SpeechServer.ini file can be used to fine-tune the operation of the Speech Server. If you alter any settings, please be sure to stop, then restart the Speech Server, for the changes to take effect.

Here are the default contents of the SpeechServer.ini file:

```
[SpeechServer]
extendedlog=true
projectpath=c:\Documents and Settings\<username>\My Documents\Character Builder Projects
```

The possible entries are as follows:

extendedlog

When set to 'true', the Speech Server maintains a log of all requests in Character Server\Logs\Extended.log.

IMPORTANT: This setting is set to 'true' by the installer. You should set this to 'false' when the solution is running smoothly, or else the extended log file will become very large.

Defaults to 'false' if not specified.

ignorecached

When set to 'true', incoming requests are always spoken, even if a request with the same text is present in the cache. This is handy during initial installation and testing. Defaults to false.

maxwavecache

Specifies the number of files to maintain in the Wave subdirectory. If an identical speech request is received (i.e. same voice and same text), and 'ignorecached' is not 'true', then the cached file is used. Defaults to 1000.

maxchars

Specifies the maximum number of characters for a speech request. The text is truncated if this length is exceeded. Use 0 to never truncate requests. This is an artificial limit designed to prevent misuse.

audioquality

Specifies the audio quality, in kbps, used by the Speech Server to compress all speech files. Valid values are 24, 32, 40, 48, 56, 64, 80, 96, 112, and 128. The default is 40, the same default used by the Character Builder.

channelN

Specifies the voices that are to run on each channel (see the following discussion). The first unspecified channel is used for all unspecified voices.

simulateportlimitations

You can set this to true with voices that are not artificially port-limited, such as Microsoft Mike and Mary, to simulate the type of port limitations present in high-quality voices such as those from NeoSpeech, Loquendo, and others.

loquendolexicon

Specifies a path to a Loquendo lexicon file that will be used for all Loquendo TTS requests.

neolexicon

Specifies a path to a Neo lexicon file that will be used for all NeoSpeech TTS requests.

map

You can remap incoming requests from one voice to another using one or more map statements. The format is:

```
map <requested voice> -> <actual voice>
```

For example the following line maps request from VW Kate to Microsoft Zira Desktop.

```
map VW Kate -> Microsoft Zira Desktop
```

Configuring Channels

The Speech Server maintains several “channels”, each served by a seperate thread. The Speech Server lets you have as many channels as you like, and assign as many voices as you like to each channel. Most TTS voices have a limited number of “ports”, e.g. a one-port TTS engine may be assigned to at most one channel. For some vendors, including Loquendo, a single port must be used for multiple voices. For other vendors, including Cepstral and NeoSpeech, each voice is independent, and can be assigned to a different channel.

The Speech Server will satisfy requests on two or more different channels concurrently, if so configured. Voices such as Microsoft Sam, Mike, Mary, and Anna are not port-limited – they can be used to generate as much audio as CPU cycles will allow. On single-core systems, splitting traffic between two ports will result in both channels operating at half-speed, however true parallelism can be achieved on multi-core systems. (Note that you can use the ‘simulateportlimitations’ setting with a Microsoft voice to make it behave like a port-limited voice, for the purposes of load testing.

Many high-quality TTS vendors artificially limit their ports so that the total amount of audio that can be generated per port is limited to the amount that can be *listened to* in real time. These “artificially limited” ports return the audio very rapidly (e.g. within 100 ms), but then make themselves unavailable for a period of time equivalent to that which it would take to listen to the audio (e.g. 10 seconds). Depending on your application (how well it caches) and your traffic level, you may therefore need to spread your requests over multiple ports.

Channels must be used in order, e.g. channel1, channel2, channel3. The first unspecified channel will be used for all voices that are not already listed. For example, the following settings would spread the NeoSpeech Kate requests between channels 1 and 2, and the NeoSpeech Paul requests between the channels 3 and 4. All remaining voices, e.g. Microsoft Mike, would go to the implicitly-specified channel 5.

```
channel1=VW Kate
channel2=VW Kate
channel3=VW Paul
channel4=VW Paul
```

The following settings would spread all requests for the Loquendo Dave and Loquendo Susan voices between channels 1 and 2, and queue all remaining requests to channel 3.

```
channel1=Loquendo Dave,Loquendo Susan
channel2=Loquendo Dave,Loquendo Susan
```

Face Server

Overview

The Face Server is a Windows service that is called by the COM API to perform PhotoFit analysis, which is very CPU intensive process. The Face Server is not started by default.

Face Analysis “jobs” are queued up and executed a few at a time, and COM API are available to monitor the execution of jobs.

FaceServer.ini

The FaceServer.ini file can be used to fine-tune the operation of the Face Server. If you alter any settings, please be sure to stop, then restart the Speech Server, for the changes to take effect.

Here are the default contents of the FaceServer.ini file:

```
[FaceServer]
extendedlog=true
```

The possible entries are as follows:

extendedlog

When set to ‘true’, the Face Server maintains a log of all requests in Character Server\Logs\Extended.log. This setting is set to ‘true’ by the installer. Defaults to ‘false’ if not specified.

avgsecs

Specifies an average compute time for each face request, in seconds, used for reporting progress. Defaults to 180.

failsafe

Specifies a maximum compute time for each face – if the request exceeds this time then the request is aborted and an error is returned. Defaults to 0 (no failsafe).

Program AB Adapter

Overview

The ProgramAB Adapter is a Windows service that is called by the Character Server when incoming requests specify an 'aiengine' value of 'programab'. In this case the 'text' parameter is interpreted not as text to be spoken, but as user input that must first be resolved to an output. That output then becomes the text to be spoken.

The details of setting up the ProgramAB adapter with the ProgramAB software are subject to change, and are described in detail at http://www.mediasemantics.com/KB/KB117.htm.

ProgramABAdapter.ini

The ProgramABServer.ini file can be used to fine-tune the operation of the AI Server. If you alter any settings, please be sure to stop, then restart the service, for the changes to take effect.

Here are the default contents of the ProgramABAdapter.ini file:

```
[ProgramABAdapter]
extendedlog=true
```

The possible entries are as follows:

extendedlog

When set to 'true', the Adapter maintains a log of all requests in Character Server\Logs\Extended.log. This setting is set to 'true' by the installer. Defaults to 'false' if not specified.

maxthreads

Specifies the maximum number of threads that will be created in order to service simultaneous requests. The default should be adequate. Though AI processing is very fast, requests can get blocked due to dependencies on outside resources. The default is 50.

javapath

Specifies the full path to the folder containing jvm.dll. If not specified, the location will be set from the current Java Developer Kit (JDK), as found in the registry.

programabpath

Specifies the full path to the Program AB folder. If not specified, ProgramAB is assumed to be installed in a child folder of the Character Server folder.

botspath

Specifies the full path to the bots folder. If not specified, the src/bots folder will be used within the Program AB folder.

ailogstore

Specifies the storage mechanism that will be used for tracking conversations. Currently the only valid value is 'mysql'.

ailogserver

If 'mysql' is used in 'ailogstore', then this parameter specifies the database used to connect to MySQL. The default is 'localhost'.

ailoguser

If ‘mysql’ is used in ‘ailogstore’, then this parameter specifies the user name used to connect to MySQL. There is no default value for this parameter.

ailogpassword

If ‘mysql’ is used in ‘ailogstore’, then this parameter specifies the password used to connect to MySQL. There is no default value for this parameter.

sessionnotifyurl

Specifies a URL that will be called by the AI Server when a “session” is judged to have completed, as specified by the ‘sessionnotifyminrequests’ and ‘sessionnotifyidleseconds’ parameters. The parameters ‘LogId’, ‘UserId’, and ‘SessionId’ are appended to this url. This is typically set to a ‘localhost’ URL, such as ‘http://localhost/SendSessionReport.asp’.

sessionnotifyminrequests

Specifies the minimum number of requests that will constitute a “conversation” for reporting purposes. Defaults to 2.

Conversation management using a database

Consider the following CharacterServer.ini entries:

```
ailogstore=mysql
ailogdatabase=ailogs
ailogserver=localhost
ailoguser=root
ailogpassword=password
```

With these settings, the Character Server will create an ‘ailogs’ database, if one is not already present, when it receives an AI request.

The database will have three tables: the Sessions table, the Variables table, and the Responses table.

Sessions Table

The Sessions table provides a record of each session (conversation). The table is read before processing each request to determine if the request is for a new user in a new session, an existing user in a new session, or an existing user in an existing session. A new entry is created in the first two cases.

logid	**userid**	**sessionid**	**time**
MyProject	User1	Session1	2010-04-14 17:14:45

Variables Table

The Variables table provides the current state of all session variables. It is read for an existing user in a new session, and written after processing each request. The value column is stored as UTF8 text.

logid	**userid**	**variable**	**value**
MyProject	User1	name	John

Responses Table

The responses table logs the actual request/response pair, and is written after each response is generated. The request and response columns are stored as UTF8 text.

logid	**userid**	**Sessionid**	**request**	**response**	**time**
MyProject	User1	Session1	my name is John	Hello John.	2010-04-14

					17:14:45
MyProject	User1	Session1	who am I	Your name is John.	2010-04-14 17:14:59

The same database is shared by all projects that use the Adapter. All tables include a logid column that is filled using the AILogId parameter provided by the client. This column allows report-generation applications to restrict a report to a given project.

Generating session emails

You can configure the Adapter to send an email each time a session ends. Consider the following entries:

```
sessionnotifyurl=http://localhost/SendSessionReport.asp'.
sessionnotifyminrequests=2
```

Here is a typical ASP mailer page:

SendSessionReport.asp

```
<%@ EnableSessionState=False %>
<%
Response.Expires = -1
Project = Request.QueryString("LogId")
User = Request.QueryString("UserId")
Session = Request.QueryString("SessionId")
Database = "ailogs"

xml = ""
xml = xml & "Project: " & Project & "<br/>"
xml = xml & "User: " & User & "<br/>"
xml = xml & "Session: " & Session & "<br/>"

dim adoConn
dim adoRS
dim counter
set adoConn = Server.CreateObject("ADODB.Connection")
set adoRS = Server.CreateObject("ADODB.Recordset")
adoConn.Open "DSN=" & Database
adoRS.ActiveConnection = adoConn
if adoConn.errors.count = 0 then
    sql = "select request, response, time from responses where logid='" & Project &
        "' and sessionid='" & Session & "' order by time;"
    adoRS.Open sql
    first = true
    while not adoRS.EOF
        if first then
            xml = xml & "Time: " & adoRS.fields(2).value & "<p/>"
            first = false
        end if
        xml = xml & "<b>User</b>: " & adoRS.fields(0).value & "<br/>" & Chr(13) & Chr(10)
        xml = xml & "<b>Agent</b>: " & adoRS.fields(1).value & "<br/>" & Chr(13) & Chr(10)
        adoRS.MoveNext
    wend

    xml = xml & "<p/>"
    xml = xml & "Current Variables for this user:<br/>"

    adoRS.Close

    sql = "select variable, value from variables where logid='" & Project &
        "' and userid='" & User & "' order by variable;"
    adoRS.Open sql
    while not adoRS.EOF
        value = Replace(adoRS.fields(1).value, """", "&quot;")
        xml = xml & adoRS.fields(0) & "=""" & value & """<br/>" & Chr(13) & Chr(10)
        adoRS.MoveNext
    wend
```

```
else
    Response.Write adoConn.Errors(0)
end if

Set Mailer = Server.CreateObject("SMTPsvg.Mailer")
Mailer.RemoteHost = "acme.com"
Mailer.Qmessage = True
Mailer.FromName = Project & " Session"
Mailer.FromAddress = "server@acme.com"
Mailer.AddRecipient "Support", "support@acme.com"
Mailer.ContentType = "text/html"
Mailer.Subject = "Session report"
Mailer.BodyText = xml
Mailer.SendMail

Response.Write "<response success=""true""/>"
%>
```

Operation & Performance

Maintaining the server

The Character Server is designed for maintenance-free operation. Once your application is running smoothly, you will want to turn off extended logging, as described in the previous sections, as the Logs/Extended.log file will otherwise grow quite large. Any errors will always be logged in the Logs/Error.log file, even when extended logging is turned off.

You should make sure that the Character Server and Speech Server services are set to start automatically. You may also want to have them restart on failure. Both settings are located in the management console under Services and Applications, Services, by right clicking on the service and selecting Properties.

Over time you will see the Flash, Wave, and Images directories grow to within 10% of the target cache size. Caches are trimmed automatically by removing the least-recently-used files.

The stress tool

The stress tool (stress.exe) simulates concurrent users of the Character Server. You normally run stress.exe from a command line. To use anything other than the default parameters, you can create a stress.ini file. Here is an example:

```
numclients=100
minsecs=5
maxsecs=30
numminutes=60
url1=http://localhost/cs/cs.exe?template=library/templates/server/speech.xml&character=da
ve&usesharedlibs=false&voice=Microsoft+Mike&text=this+is+request+UNIQUE.
url2=http://localhost/cs/cs.exe?template=library/templates/server/speech.xml&character=ma
ry&usesharedlibs=false&voice=Microsoft+Mary&text=this+is+request+UNIQUE.
url3=http://localhost/cs/cs.exe?template=library/templates/server/speech.xml&character=ja
mes&usesharedlibs=false&voice=Microsoft+Mike&text=this+is+not+a+unique+request.
```

With this stress profile, 100 clients are pulling at random from the 3 urls specified, at random intervals between 5 and 30 seconds apart, for a period of one hour. The string “UNIQUE” is replaced with a pseudorandom number, so as to form a realistic work load in the face of the built-in TTS and Flash cache mechanisms.

Speech licensing and throughput

Character Server features such as dynamic Flash generation and question-answering can require considerable server-side resources. The total number of simultaneous clients that can be served by a single Character Server depends on many factors, but the gating factor tends to be the Text-to-Speech (TTS) engine.

TTS requests are normally queued up, and the actual TTS generation occurs at several times real-time. For example, going from a text sentence to a flash stream consisting of a character speaking that sentence may require 100 milliseconds of intense CPU activity on a commodity server. During this time the audio is generated, compressed, and inserted into the Flash stream, along with animation frames selected from the character's animation library. Additional requests are queued while a request is being processed, which will affect the latency, i.e. the total time between when a request is made and a response is received. In many cases the Server will improve the overall throughput by using a cache to detect identical requests and redirect these to previously-created Flash files.

The Character Server is multithreaded to take advantage of multicore CPUs, however Text-to-Speech licenses tend to be sold “per port”. Depending on your application, your load, and your Text-to-Speech engine, you may need multiple Text-to-Speech ports in order to achieve the maximum throughput on even

a single core CPU. The Character Server's Speech Server component can be configured to take full advantage of speech products by Loquendo, NeoSpeech, Cepstral, and others. For more details, please see the 'SpeechServer.ini' section.

Character licensing and throughput

Characters and addon packages have per-user licenses. Unlike the Character Builder, the Character Server will normally service several users at the same time. You can purchase stock characters for use with the Character Server in the same manner as for the Character Builder. However, as with channel-based speech products, character products are normally limited so that they can only be accessed by one Character Server user at a time.

For example, if user A requests a stream using the "Al" character, and the request results in 120 frames (10 seconds) worth of Al animation, then the Al license will be unavailable to other users for generation for 10 seconds following the request, even though the request itself is serviced very rapidly. If a user B requests a new resource using Al during this time, then the request will be delayed until the license is freed up, in this case up to 10 seconds if user B's request arrived shortly after user A's request. Effectively a single user of an online system would never see delays, since the character license remains in use for precisely the period of time required to view the previous generation. Requests that are returned directly from the cache do not require generation and do not use up a character license. Similarly custom characters created by either Media Semantics or users of the CBEK are not affected by character license pooling.

As you move beyond the test phase, you will want to purchase multiple instances of a character license and request that they be assigned to a Character Server license to form a license pool. For example, with a pool of two "Al" licenses, the chance of delay is reduced substantially, since up to two users can be generating concurrently. License pooling is enabled in Character Server 4.4.2 or higher. Additional character licenses can be purchased from the purchase page at the regular price, and can be assigned to a Character Server license after your purchase by placing a request with our support team. The request will result in an updated verification.txt file which you will install in your Character Server directory. Character pack products such as the Realistic Pack can also be assigned to a pool. For example a pool of two Realistic packs is equivalent to a pool of two Steve characters, a pool of two Robert characters, etc.

The right pool size depends on many factors – you can use the Character Server's Stress tool to simulate a traffic load, and monitor warnings in the Extended log file that appear when one or more users are waiting on a particular resource. The first few characters that you add to a character pool will have the most impact on performance, with the benefit tapering off as the number of characters in the pool increases. A pool size of 10 is considered equivalent to an infinite pool size by the Character Server, which will no longer attempt to meter the usage for that character.

Securing the server

After a Character Server solution is running smoothly, there are a few things you might want to do to secure your server against malicious attacks.

We will examine three types of threat: Hacking, Scamming, and Denial of Service.

Hacking

By Hacking we mean any activity that would result in sensitive information being read or altered through the operation of the Character Server API.

If you are using the COM API then note that this API is never exposed directly – it is only exposed through your server pages. You may already be familiar with procedures to harden ASP, ASPX, or PHP pages against attacks, such as avoiding any arguments that may get executed on the server. It is possible to pass XML actions inside a text parameter, however the resulting actions are embedded in the Flash file, so they are run on the client-side, and never on the server side. The AI Server does not let you pass in new rules for

execution, but the rules already present on the server include actions that can collect information via http requests, however these http requests are always run within the context of the anonymous http user.

If you are using the CGI interface (i.e. you are using Character Server Standard), then the design of this API is such that it is not possible to alter any information outside of the Character Server's cache subdirectories and AI Server database, or read any information outside of the Library, Projects, or Data paths, as set up in the ini files.

Scamming

By Scamming we mean any activity in which a third party uses Character Server resources on your server without paying for them.

The 'same-domain' rule makes it impractical for someone to create a Character Builder application and simply point the External Presentation to your server instead of their own, unless you allow users to publish flash files to your server.

A more savvy user could discover the URL to your cs.exe (the CGI API), and pass it requests. One deterrent to doing this is that the Flash streams generated by cs.exe are difficult (but not impossible) to consume by themselves. Remember that any request consists of the name of a template and values to fill into this template. This type of scamming is somewhat facilitated by the fact that the Character Server installs the CGI interface to a default location (http://server/cs/cs.exe), and offers a very general standard template (Speech.xml) as part of the default installation. To protect yourself against potential scamming attacks, you might consider renaming the cs.exe url from the default one used by the installer. However remember that anyone viewing your application under a packet sniffer would be able to see the actual cs.exe url you are using, so this is not a very strong protection. A slightly stronger protection would be to avoid using Speech.xml as a template, in favor of a template that hardcodes important information, such as the character you are using, thereby making it less attractive. More sophisticated gatekeeping can be done by avoiding the CGI interface altogether, and making all requests go through the COM interface – this then allows you to require the presence of a session cookie that you have written by some other means.

A more pervasive kind of scamming, and one that is difficult to protect against, is the practice of running your Flash file on a different site, using a Flash OBJECT/embed tag with an absolute reference. This is actually a general problem shared by all Flash files, especially Flash games, and it is especially tempting for scammers if you offer a generic application, such as a general purpose chat bot, translation bot, or pronunciation bot. There are solutions to this problem at the web server that involve filtering requests to flash files so that they are only made available to calling pages from the same domain. However the simplest way to protect yourself from this kind of attack is to create an application that is unattractive to scam because it contains your logo, talks about your organization, etc.

Denial of Service

By Denial of Service we mean any malicious activity that consumes resources to the point of denying them to your regular users. The Denial of Service problem is similar to the Scamming problem, except that the attacker is not interested in the actual resources produced, so it does not help to "poison" them to make them less generally useful. Such attacks are typically less of a threat only because there is often little motivation to carry them out, but they are also harder to deter. Traditionally applications look at the IP address of a requester and deny an overwhelming volume of requests coming from one IP. This forces attackers to use multiple computers to perform the attack successfully. The CGI interface does not, itself, offer IP-based Denial of Service protection, so any such protection would need to be implemented using a lower-level mechanism.

Appendix A – COM API

Creation

The Character Server COM API can be created by its class identifier “CharacterServer.CharacterServer”. The actual syntax varies from one environment to another. For example you can create a JavaScript command script for execution with wscript.exe: simply create a file with an extension .js and you can then double-click it to run it directly from the Windows shell. In a command script you would use the following syntax to create the COM object:

```
cs = WScript.CreateObject("CharacterServer.CharacterServer");
```

Alternatively, when running within an ASP file within IIS, you would use the following syntax:

```
cs = Server.CreateObject("CharacterServer.CharacterServer");
```

In PHP, the syntax is:

```
$cs = new COM("CharacterServer.CharacterServer")
```

Additional scripting language examples can be found in the Character Server's ReadMe file.

Supporting methods

SetBasePath(String path)

Specifies full path for directory at which the source XML string would be located if it were a file, for use in resolving relative paths in XML.

SetLibraryPath(String path)

Sets the library path for this session. The library path specifies the physical location of the Library directory, and is used to resolve references of the form "Library/xyz". If a library path is not specified, then the 'librarypath' entry of the CharacterServer.ini file is used.

SetDataPath(String path)

Sets the data path for this session. The data path specifies the physical location of the Data directory, and is used to resolve references of the form "Data/xyz". If a data path is not specified, then the 'datapath' entry of the CharacterServer.ini file is used.

SetParameter(String name, String value)

Sets the value of a template parameter. The XML string in each of the Render API can be either an actual XML message or a filename. If a filename is used, then this file is used as a template file. As with the CGI interface, the template file is first read and then parameters are replaced with their values. Parameter values are marked with identifiers beginning with the '$' sign. Use the '$$' sequence to represent an actual '$' symbol in the content. The default value for an unspecified parameter is a blank string.

SetTimeout(Integer seconds)

Specifies a timeout, in seconds, to be used on requests to the Character Server. By default the timeout is approximately one minute. Set the Timeout to 0 to never time out.

out = ConvertUTF8(String in)

The COM API is UTF-16, however XML files are normally stored as UTF-8. If you are reading your own template file then you can use this API to convert it to UTF-16.

out = ConvertUTF16(String in)

This API converts a 16 bit string to a UTF-8 string. While all COM API return 16 bit wide strings, the upper byte of each character is always 0 and the lower byte contains the proper UTF-8 encoding, so that you can write it to a UTF-8 file or stream.

Rendering

RenderAsResponse(String xml)

Renders an XML message directly to the IIS response object. The 'xml' parameter can be either an XML string or the name of a template xml file. This API uses the Flash cache. Under IIS, renders an XML string as a Flash SWF stream, and inserts the stream directly to the response object. This API returns an error under all other environments.

In many cases you will use RenderAsResponse (or RenderToTempFile) as an alternative to the CGI API when you need more control over the processing. For example you might want to check user credentials or insert your own query processing, before passing the request onto the Character Server. To do this, you can follow this sequence (shown with ASP syntax):

```
text = Request.QueryString("Text")
' additional text processing as necessary
cs = Server.CreateObject("CharacterServer.CharacterServer")
cs.SetParameter("Text", text)
cs.SetParameter "Character", Request.QueryString("Character")
cs.SetParameter "Voice", Request.QueryString("Voice")
cs.RenderAsResponse("Library/Templates/Server/Speech.xml")
```

String file = RenderToTempFile(String xml)

Renders an XML message to an SWF file. The ‘xml’ parameter can be either an XML string or the name of a template xml file. After rendering to a file, you can subsequently redirect the request to that file. This API uses the Flash cache for the output file. You typically use RenderToTempFile as an alternative to RenderAsResponse, which is only available under IIS. Simply follow a RenderToTempFile with a redirect to the generated file.

String file = Render(String xml, String path)

Renders an XML message to an SWF file. The ‘xml’ parameter can be either an XML string or the name of a template xml file. The ‘path’ parameter can be a specific file, e.g. “test.swf” or a directory. If a directory is specified, then the method will create a unique file in that directory and then return the unique file name. The resulting SWF file is not cached by the Character Server.

The CGI interface permits output without a loader, i.e. loader="external", and is normally used to generate flash streams that are consumed by a Character Builder project using the External File mechanism. In contrast, the Render API is used to publish arbitrary scenes that are specified by instantiating a template or by assembling an XML string from first principles. In this mode you can use loader="standard" to access the complete capabilities of the Character Builder application model, including smooth-switching between messages and idles, embedded flash movies, slideshows, etc.

The Builder is used simply as a template-building or reference tool in this case.

String file = RenderVideo(String xml, String path, String format, String params)

Renders an XML string to a video file. The ‘xml’ parameter can be either an XML string or the name of a template xml file. The path parameter can be a specific file, e.g. “test.avi” or a directory. If a directory is specified, then the method will create a unique file in that directory and then return the unique file name.

The Character Server always produces uncompressed AVI as a first step, but it will call any command-line executable as a post step to convert the avi to compressed formats, including 3gp, mov, wmv, and flv. To use post-avi compression, the ‘format’ string should be set to the desired format, i.e. 3gp, mov, or flv. If not specified, the extension of the file specified by the ‘path’ parameter is used. Use the “avi” format to specify no post compression. The CharacterServer.ini file’s ‘3gp’, ‘mov’, ‘wmv’, 'jpg', and ‘flv’ parameters can be used to register a suitable compressor with the Character Server. The ‘params’ argument is inserted as the $params value, and may be left blank.

The default value for all parameters uses ffmeg.exe, a binary distribution of the open source ffmpeg tool included with the Character Server package for your convenience.

You can run this executable from the command line to discover its many options, or substitute your own command-line compression tool by setting your own CharacterServer.ini values. Note that Character Server always adds a *-r* parameter with the correct project framerate, as this is always necessary to compensate for a fixed frame-rate internal format used as an intermediate form in the generation.

String file = CompressVideo(String infile, String outfile, String params)

CompressVideo() is often used as a follow up to RenderVideo() to transcode the video to another format. For example you can get a jpeg thumbnail for your MP4 video by calling CompressVideo("My.mpr", "My.jpg", "").

Async render

GetAsyncRenderQueueLength()

Returns the number of pending Render requests for the server.

String result = StartAsyncRender(String xml, String path, Integer Deadman)

The operation of this API is similar to Render(), execept that a "request token" is returned that can be used with the CheckRender() and CloseRender() API. The Deadman parameter is reserved and should be 0.

CloseAsyncRender(String request)

Closes an async request and cancels any processing that may still be ongoing for the request.

String result = CheckAsyncRender(String request)

Returns "INVALID", "INCOMPLETE", "COMPLETE", or "ERROR". After launching an asynchronous Render with StartAsyncRender(), the caller should periodically call CheckAsyncRender() to check the status of the request. The "INVALID" code indicates that the request is not, or no longer is valid. For example a request is no longer valid after it has been Closed. The "ERROR" code indicates that there was an irrecoverable error in the render.

AI processing

SetAIEngine(String engine)

Specifies an AI Engine for use by the Respond API. Acceptable values are "programab".

SetAIDatabase(String file)

Specifies an AI Database for use by the Respond API.

SetAIVariables(String in)

Sets AI variables for use during this session. This method can be used to save and restore conversation state between invocations. AI variables are stored in uuencoded format, for example

"name=John&gender=male".

If not specified, AI variables will be loaded for the current user using a database or XML files (see AI Server).

SetAILogId(String id)

If specified, the Server will use this identifier to maintain conversation state using an external database, as described in the AI Server section.

SetAIUser(String user)

If specified, the Server will use this string to maintain conversation state, as described in the AI Server section.

SetAISession(String session)

If specified, the Server will use this string to maintain conversation state, as described in the AI Server section.

String out = Respond(String in)

Returns a string output from a string input. You must have loaded a file using the Load method prior to calling Respond.

String out = GetAIVariables()

Returns all AI variables as a string in MIME format. This method can be used to save and restore conversation state between invocations. AI variables are stored in uuencoded format, for example

"name=John&gender=male"

error = Compile(String file)

Compiles an XML file to the MSF format. If a .msf file is present, and it is more recent than the .xml file with the same name, then this .msf file will be loaded instead, for greater performance. The empty string is returned if there is no error, and an error message is returned in the case of a syntax error.

Audio processing

SpeakToFile(String text, String voice, String file)

Speaks the given text to a file using a given voice. The Server must have write permission to the file for this operation to succeed.

LipSync(String file, String text, Integer model)

Ensures that the give .wav file is lip-synced. The 'text' parameter should be a text-transcription of the audio file contents. The model parameter can be 1 (low accuracy), 2 (medium accuracy), or 3 (high accuracy). The Server must have write permission to the file for this operation to succeed.

Compress(String in, String out, Integer quality)

Produces an MP3-compressed file 'out' from an uncompressed file 'in'. The 'out' file can be the same as the 'in' file. The quality parameter is ignored, and should be set to 0. The audio compression quality is fixed, and is determined by the 'audioquality' parameter in the SpeechServer.ini file.

DecodeFLV(String in, String out, String params)

Produces a wav-compressed file 'out' from an flv audio file 'in'. The params argument should normally be an empty string. The Character Server calls out to a command-line executable such as 'ffmpeg' to perform the decompression. The command-line utility is registered with the Character Server using the CharacterServer.ini file's 'decodeflv' parameter.

The default value for 'decodeflv' uses ffmpeg.exe, a binary distribution of the open source ffmpeg tool included with the Character Server package for your convenience:

ffmpeg.exe -i $infile $params $outfile

You can substitute your own command-line compression tool by setting your own CharacterServer.ini 'decodeflv' value.

Face design

CreateDefaultFace(String controlfile, String facefile)

Sets the file 'facefile' to the default (i.e. "factory setting") face. The 'controlfile' parameter is passed to most Face Design API, and is normally set to "Library\models\control\si.ctl".

CreateRandomFace(String controlfile, String facefile, String race, Boolean lockgender, Float gendershape, Float gendertexture, Boolean lockage, Float ageshape, Float agetexture, Boolean lockcaricature, Float caricatureshape, Float caricaturetexture, Boolean lockasymmetry, Float asymmetry)

Sets the file 'facefile' to a random face. The 'race' parameter can be one of "all", "african", "european", "seasian", and "eindian", and lets you lock in a specific race. The remaining parameters let you lock down one or more of the broad parameters Gender, Shape, Caricature, and Asymmetry.

The acceptable ranges for these parameter are as specified in the respective Set API, e.g. SetGender().

SetGender(String controlfile, String facefile, String race, String type, Float gender)

Sets the gender in the file 'facefile'. In all Get and Set API, the race parameter can be one of "all", "african", "european", "seasian", and "eindian", and indicates the statistical race model to be used. For example this allows a South East Asian face to be altered differently than an African face with regards to Age and Gender. In most cases the "all" parameter can be used as the differences between races are relatively minor. The type parameter in all Get/Set API can be one of "shape" or "texture", and indicates which aspect of the face to change. The gender parameter is a value between -4 and 4, with -4 indicating very male and 4 indicating very female.

Float gender = GetGender(String controlfile, String facefile, String race, String type)

Gets the gender parameter from the file 'facefile'.

SetAge(String controlfile, String facefile, String race, String type, Float age)

Sets the age of a face. The 'age' parameter is specified in years and should normally be in the range of 15 to 65.

Float age = GetAge(String controlfile, String facefile, String race, String type)

Gets the age parameter from the file 'facefile'.

SetCaricature(String controlfile, String facefile, String race, String type, Float caricature)

Sets the caricature level of a face, that is the degree of emphasis of the differences from the norm for symmetric shape and texture controls. The 'caricature' parameter is a floating point value between 0 and 2, with 0 indicating no difference, 1 indicating typical difference, and 2 indicating extreme difference.

Float caricature = GetCaricature(String controlfile, String facefile, String race, String type)

Gets the caricature parameter from the file 'facefile'.

SetAsymmetry(String controlfile, String facefile, String race, String type, Float asymmetry)

Sets the asymmetry level of a face, that is the degree of emphasis of the differences from the norm for asymmetric shape parameters. The 'caricature' parameter is a floating point value between 0 and 2, with 0 indicating no difference, 1 indicating typical difference, and 2 indicating extreme difference.

Float asymmetry = GetAsymmetry(String controlfile, String facefile, String race, String type)

Gets the asymmetry parameter from the file 'facefile'.

SetRaceMorph(String controlfile, String facefile, String racefrom, String raceto, Float tween)

Adjusts a face to look more like one race or another. The 'racefrom' and 'raceto' parameters can be one of "all", "african", "european", "seasian", and "eindian". The 'tween' parameter should be between -1 and 1, with -1 indicating a result identical to 'racefrom' and 1 indicating a result identical to 'raceto'.

Float tween = GetRaceMorph(String controlfile, String facefile, String racefrom, String raceto)

Gets the degree to which the face is morphed between any two races.

Integer n = GetControlCount(String controlfile, String type, String sym)

Returns a count of the controls of the given 'type' ("shape" or "texture") and 'sym' value ("symmetric" or "asymmetric").

String name = GetControlName(String controlfile, String type, String sym, Integer n)

Returns an English-language name for the given control. The 'n' parameter must be in the range 0 to GetControlCount()-1.

SetControlValue(String controlfile, String facefile, String type, String sym, Integer n, Float value)

Sets the given control value. The 'n' parameter must be in the range 0 to GetControlCount()-1. The 'value' parameter must be in the range -10 to 10, with 0 indicating the mean value.

Float value = GetControlValue(String controlfile, String facefile, String type, String sym, Integer n)

Returns the given control value. The ‘n’ parameter must be in the range 0 to GetControlCount()-1.

Photofit

String hex = ReturnFaceAsHex(String facefile)

Returns the contents of the given face file as a hexadecimal string.

CreateFaceFromHex(String facefile, String hex)

Creates a face file from a hex string as produced by ReturnFaceAsHex().

GetPhotoFitQueueLength()

Returns the number of pending PhotoFit conversions for the server.

String result = DoPhotoFit(String controlfile, String image1, String pt1, String image2, String pt2, String image3, String pt3, Boolean removehair, Boolean detailfromside, String resultfile)

Performs a synchronous PhotoFit, i.e. returns only when the PhotoFit is complete. Up to 3 images may be specified - if an image is not specified then a blank string should be passed for that file. If only a front image is used then this should be passed as ‘image1’. If a front and a left profile image are specified then these should be passed as ‘image1’ and ‘image2’ respectively. Finally, if a front, left, and right profile image are specified then these should be passed as ‘image1’, ‘image2’, and ‘image3’ respectively. Image files must be in the JPEG format, and can consist of a full path to a local file, or a URL to a file that may be located on another server. For each specified image, a corresponding ‘pt’ string must also be specified with the form "x1,y1,x2,y2,...,xn,yn", with the control points for each image. The front image has 11 control points, while the optional profile images have 9 control points each. The 'removehair' parameter is normally false, but it can be set to true to remove any facial hair from the texture detail. The 'detailfromside' parameter is normally false, but it can be set to true to pull face detailing information from both the left and right profile images. The 'resultfile' parameter must be set to a valid filename for use in writing the resulting file, or to an empty string. If a file is specified then this same file is returned as the result of the API. If an empty string is specified then the result is a hex string suitable for use with CreateFaceFromHex().

String request = StartPhotoFit(String controlfile, String image1, String pt1, String image2, String pt2, String image3, String pt3, Boolean removehair, Boolean detailfromside, String resultfile, Integer Deadman)

Starts an asynchronous PhotoFit. The operation of this API is similar to DoPhotoFit(), execept that a "request token" is returned that can be used with the CheckPhotoFit() and ClosePhotoFit() API. The Deadman parameter is reserved and should be 0.

ClosePhotoFit(String request)

Closes a PhotoFit request and cancels any processing that may still be ongoing for the request.

String result = CheckPhotoFit(String request)

Returns "INVALID", "ERROR", "INCOMPLETE", "INCOMPLETE (N pending)", "INCOMPLETE (N%)", or a filename or hex string representing the result of the analysis. After launching an asynchronous PhotoFit with StartPhotoFit(), the caller should periodically call CheckPhotoFit() to check the status of the request. The "INVALID" code indicates that the request is not, or no longer is valid. For example a request

is no longer valid after it has been Closed. The "ERROR" code indicates that there was an error in the PhotoFit process, and that the user should try again with a different image or control points.

Photomorph

String size = ProcessPhoto(String infile, Integer scale, Integer cropleft, Integer cropright, Integer croptop, Integer cropbottom, Integer flags, String outfile)

Processes a JPEG, PNG, or BMP format file ‘infile’ to provide scaling, cropping, and background removal. The result is written to the file ‘outfile’. Both ‘infile’ and ‘outfile’ should be fully-qualified file system paths, and ‘outfile’ must have the ‘.png’ extension. The ‘scale’ parameter is an integer in the range 1-100. The following flags are available:

1 = Perform “green screen” removal
2 = Include a checkered background to indicate transparent areas

These values may be added together.

String params = DoPhotoMorphAnalysis(String basefile, String facefile, String photofile, String pt)

Given a reference to a PhotoMorph animation library, a .fg face model file, a PNG file as processed by ProcessPhoto, and a string of points, produces a list of parameters ready for insertion into the ‘character’ element, such as:

```
facescale="0.55" pixperunit="0.20"
```

The format of the point string is the same as that used by StartPhotoFit, however the points should be relative to the processed PNG file and not the original JPEG file, so they should take into account any scaling and any cropping on the left and top edges.