Project ref. no.	*EDC-22124 ESIGN / 27960*
Project acronym	**eSIGN**
Project full title	**Essential sign language information on government networks**

Security (distribution level)	PP
Contractual date of delivery	30 June 2004
Actual date of delivery	18 September 2004
Deliverable number	D2.6
Deliverable name	Tools Integration with Portal Systems
Type	Report
Status & version	Final
Number of pages	14 plus Appendix
WP contributing to the deliverable	WP 2
WP / Task responsible	UEA
Other contributors	Viataal, SI
Author(s)	John Glauert and Judy Tryggvason, UEA Contributions by Margriet Verlinden, Viataal, and Uwe Ehrhardt, SI.
EC Project Officer	Evangelia Markidou
Keywords	Avatar, Virtual Human, Animation, Sign Language, eGovernment, Portals, Tools Integration
Abstract (for dissemination)	The deliverable reviews the different approaches to content creation that have been used in the course of the eSIGN project. Tools are described that have been used to produce content for websites and other applications with the involvement of third parties such as web authors and designers. Classes of content required on signed web sites are considered and appropriate workflow described. There is a brief review of eSIGN content in progress and a discussion of how content creation can be integrated into the content management processes for the portals in use. The deliverable incorporates and expands material provided in Milestone M2.4 *Specification of workflow for signed content creation with CMS.*

1 EXECUTIVE SUMMARY 4

eSIGN Tools 4

Content Management Systems 4

Content Styles 4

Creating Content for Deaf Users 4

Portal Integration for eSIGN Portals 4

2 A SURVEY OF ESIGN TOOLS 6

2.1 Introduction 6

2.2 The eSIGN Editor 6

2.3 The Structured Content Generation Tool 7

2.4 The Web Page Cookbook 7

2.5 The VANESSA Phrase Editor 8

3 CURRENT TOOL USE FOR CONTENT CREATION 8

3.1 Introduction 8

3.2 The German Site 8

3.3 The British Site and VANESSA 9

3.4 The Dutch Site 9

4 A SURVEY OF EXISTING CMS AND WEB DEVELOPMENT PROCESSES 10

4.1 Introduction 10

4.2 Content Management for the German Portal 10

4.3 Content Management on British Sites 10

4.4 The Viataal approach 10

5 CONTENT STYLES 10

5.1 Introduction 10

5.2 Free form text 11

5.3 Forms 11

5.4 Structured content 11

6 CONTENT CREATION FOR DEAF USERS 12

6.1 Introduction 12

6.2 Handling multilingual content 12

6.3 The particular challenges of sign language content 12

6.4 Linking the Editor to Content Management Systems 12

7 PORTAL INTEGRATION IN ESIGN 13

7.1 Introduction 13

7.2 Integration with the German Portal 13

7.3 Integration with the British Portal 13

7.4 Integration with the Dutch Portal 13

1 Executive Summary

eSIGN Tools

A number of Tools have been developed during the course of the eSIGN project to assist in the creation of signed web pages, and a sign language translation system.

The animation and display of eSIGN content depends on the avatar signing components developed by the University of East Anglia and Televirtual Limited. The primary content creation Tool is the eSIGN Editor, developed by the University of Hamburg.

The Structured Content Generation Tool (SCGT) caters for web pages where the format of the pages remains consistent. For pages where the content is not so structured, a Cookbook has been developed to act as a manual to help users to add signed content.

Content Management Systems

Content developed using eSIGN tools will become out of date quickly unless working practices keep signed content updated in line with changes to text pages. It was anticipated that maintainers of these websites would already be using content management systems.

A survey has been made to understand current practice of eSIGN partners in order to develop processes that would make it straightforward to keep signed content up to date.

Content Styles

All web pages including eSIGN content will need to include scripting to launch the avatar or link to a window or pane already containing the avatar. Scripts will send SiGML data to the avatar at the appropriate time, often in response to requests from users via buttons or hyperlinks.

Some pages are translations of pages with free-form information and provide signed versions of some or all of the textual material.

Where signing support is added to web forms, the approach is to provide signed help and explanation to assist deaf people in completing forms.

For pages that present structured information on related topics, as in the job vacancies listings on the Viataal site, content is generated using an application of the SCGT.

Creating Content for Deaf Users

Since signed material produced using eSIGN tools appears as conventional XML text in SiGML, many aspects of content creation are the same whether the output is for hearing or deaf people. The report discusses special techniques required to design sites that present content adapted to the needs of deaf people and accommodate the avatar used to display signing.

Portal Integration for eSIGN Portals

When eSIGN was planned, it was envisaged that those managing eGovernment portals would use sophisticated content management systems to deliver information to clients. Since the project would focus on creating example content using specially developed tools, there would be a danger that eSIGN content would not be updated along with content for hearing users and would rapidly become obsolete.

The need was identified to consider links between eSIGN tools and portal software in order to provide eGovernment content creators with a practical means of including signed content along with multilingual content and content for users with special needs. In the event, the portals used by those deploying eSIGN content are less sophisticated than was envisaged so the need for linking is only now becoming important. The current state and future plans for eSIGN Portals are discussed.

2 A Survey of eSIGN Tools

2.1 Introduction

A number of Tools have been developed during the course of the eSIGN project to assist in the creation of signed web pages, and a sign language translation system.

The animation and display of eSIGN content depends on the avatar signing components developed by the University of East Anglia and Televirtual Limited. An Active-X component has been developed that takes signed content encoded in SiGML (Signing Gesture Markup Language), generates animation data for one of a number of avatars, and animates the signed content in a web page or other Windows application. See Deliverable D2.1.

The primary content creation Tool is the Editor, developed by the University of Hamburg. This facilitates the creation of sign animation for the avatar, from HamNoSys descriptions of sign language translations of text, or additional explanatory information. Animations may also be constructed from previously created signs stored in a lexicon.

Once signed content has been created, the process by which it is added to web pages depends to a large extent on the nature of the original content. The Structured Content Generation Tool caters for web pages where the format of the pages remains consistent. For pages where the content is not so structured, a Cookbook has been developed to act as a manual to help users to add signed content. A variety of features and formats are available for selection. Finally, the VANESSA system includes an application which allows new translations to be added not only in any sign language, but in other foreign languages as well.

2.2 The eSIGN Editor

The eSIGN Editor software allows the user to compose signed text to be performed by the eSIGN Avatar. The editor gives the user an economic approach to creating signed sequences by selecting signs from a lexicon (database) and then modifying them with the assistance of specialised editors (focussing on particular aspects of the sign's phonetics and morphology) where necessary. Mouth gestures, facial expressions and locations are managed in this way. Where a sign is not already in the lexicon, it may be added by specifying the HamNoSys string which describes it. A detailed description of the Editor is given as Deliverable D2.3. A good overview of the Editor's components is given in diagrammatic form as Figure 3 in the dissemination document, *The eSIGN Approach*, published as Deliverable D7.2.

The Editor is available both in versions for PCs and Macs, to cater for the differing requirements of eSIGN partners. While it is possible to describe most signs using the Editor, it is possible to add plug-ins in the form of specific editors for each of the target sign languages, for example for fingerspelling.

The creation of sign sequences is facilitated not only by a clear interface to the lexicon, but also by Avatar playback controls which allow sequences to be reviewed as part of the development process.

Signs are described in the lexicon by glosses, spoken-language labels for signs that match the semantics of a sign as closely as possible. Import and export to the lexicon is possible. Import is from text files in a format specified in the Appendix to D2.3. Export is in the form of SiGML which can then be integrated into web pages and other applications.

2.3 The Structured Content Generation Tool

Developed to allow input by users with only limited technical expertise, the Structured Content Generation Tool (SCGT) is useful for web pages where the structure of the information, and thus of the web page itself, remains fairly constant, although the information itself may change frequently. Details of the SCGT are given in Deliverable D2.4. The basic workflow is described in Figure 1 below.

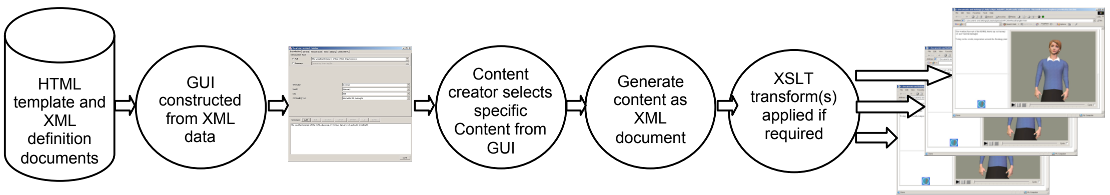

Figure 1

Two applications were developed to demonstrate the Tool's use; a Weather Forecast Creator and a Flight Timetable Creator. It was hoped that using these examples would be a sufficient guide for the development of other similar Tools. However, it did require a reasonable degree of technical expertise. On reflection, it would have been better to develop a further Tool (an SCGT Creator) to make the building of new SCGT's easier. Content creation tools that could be created by users who were to use them would have been a significant advantage. An SCGT Creator could be written using a similar process to that used by the SCGT itself. Unfortunately, lack of time prevented development of an SCGT Creator tool within the scope of the eSIGN project.

SCGT applications have a number of advantages:

- No re-coding of the core Tool itself is required when developing specific applications for new websites.
- All applications that use the Tool will only differ in three ways; the XML input they require, an (optional) XSLT stylesheet, which can be used to refine or restructure the Tool's output XML before the final HTML is generated, and the HTML template page. Thus to develop a new application, only these files will need to be altered.
- The entire Graphic User Interface (GUI) of any application which uses this Tool is also specified as part of the XML input. The input interface used to build the sign sequences can therefore vary according to the kind of information output structure that is required.
- The Tool is written in Java and is thus available for use on any platform which has the Java JRE or JDK installed.
- Creating new web pages using the Tool is extremely fast.
- Web pages can be created in up to three different sign language versions simultaneously.

2.4 The Web Page Cookbook

The majority of web pages, however, are not of the kind for which the SCGT is appropriate. A more flexible approach was required, and the Web Page Cookbook was developed for this purpose. The basis of the Cookbook was Milestone M2.2, the

Form-Driven Transaction Support Toolkit. This has now evolved as an amalgamation of the experience gained as all three of the partner websites were developed. The Cookbook is included here as an Appendix.

Sections of the Cookbook were used as a guide both by the developer of the British site, and partners at Viataal in the Netherlands, to aid web page development. The completed Cookbook starts with the most basic web page with signing, and then describes the way in which additional components, such as hyperlinking the text itself, controls for enhanced avatar viewing, and alternative ways of adding the SiGML to be used for avatar animation, can be added.

Web page development of this kind is very much piecemeal and would not blend easily with any CMS, where a more structured environment is required. However a Tool to facilitate such maintenance would be feasible.

2.5 The VANESSA Phrase Editor

A Tool was developed as part of the VANESSA application suite to maintain content for the system, which is fully described in Deliverables D2.5 and D4.2. Phrases may be added to the system in one or more languages. The Tool ensures that at least one written phrase, alternative ways of saying that phrase (to enhance text or speech recognition), and one translation are added as part of a single maintenance operation. While this Editor differs in that it is not concerned with content available through portals, it does have many similar features, notably the structured nature of its signed content. An additional aspect is that VANESSA is easily modified to translate into other spoken languages as well as sign languages.

The VANESSA system was designed for operation in Help Desk environments. However, as communication between Clerk and Client computers is across a network link in the form of XML, there would be nothing to prevent the system being used for interactive online help on one kind or another.

3 Current Tool Use for Content Creation

3.1 Introduction

In the sites chosen as the three eSIGN demonstration sites, modifying the design of existing pages was one of the first problems which had to be to be addressed. This is discussed further in Section 5. What became apparent was that the initial modification of existing pages was not trivial, nor was there an appropriate Tool (beyond standard web page editors such as Macromedia Dreamweaver or Adobe GoLive) to deal with the particular requirements necessary to add signed content. Obvious contenders for further research are stylesheets containing all the signing information, which could be applied to web pages if required.

However, the Tools developed by eSIGN partners were of considerable use in the course of web page creation.

3.2 The German Site

Three sections of the hamburg.de website were selected for the addition of a signed translation of web page text:

- Welcome and the Integration Office ("Integrationsamt") group of information site parts
- Information about the Hamburg Central Lost Property Office
- Hamburg City Parliament

Content creation for these pages is described in Deliverables D3.1, D3.2 and D3.3 respectively.

eSIGN German partners created content with the eSIGN Editor (Section 2.2). Content thus created was added to modified pages manually. However, they did recognise that as the content area grew, the effort involved with creating and updating static pages would become cumbersome. The content would then be better made available from a centralised database. Content could then be generated and presented in a dynamic fashion. At that point, effort would shift from maintaining up-to-date static pages to maintaining actualised content in the database for dynamic presentation (see Deliverable D3.1).

The editors of hamburg.de currently use content management systems (e.g. CoreMedia or Vignette). The German eSIGN partners envisaged that editors/translators of the content for Deaf people would use the same workflow.

3.3 The British Site and VANESSA

For FormSign and the Deaf Connexions web pages (Deliverable D4.1) the use of Tools was restricted to the eSIGN Editor (Section 2.2) and some assistance from what has become the Web Page Cookbook. As the amount of signed content was relatively small, even with the additional signed explanations to help with form-filling, editing the original text pages manually was an achievable objective. SiGML created in the editor was added to pages as required. No consideration was given to how the site might be maintained in the future beyond the observation that it was possible for the SiGML required by any page to be stored in separate files for ease of maintenance.

Although phrases for VANESSA could have been added to the system using the VANESSA tool, delays meant that in the event signed phrases were added directly to the appropriate directory, and added by hand to the text file which referenced text/spoken phrases with their signed equivalents. While it is possible for a VANESSA system to expand as additional phrases become available, and the VANESSA tool could be used to facilitate this, VANESSA differs from web page content management in that it is unlikely that phrases will be amended or deleted from a particular installation. Nonetheless, the tool is able to perform such maintenance if required.

3.4 The Dutch Site

The Arbeidscentrum Viataal (Viataal Job Centre) pages afforded opportunities to use the SCGT (see Section 2.3) most effectively. The Web Page Cookbook also came about partly as a result of enquiries from the Dutch team. As with the other sites, content was created using the eSIGN Editor (Section 2.2), together with particular plugins for fingerspelling, etc. Details of all the tools used are given as part of Deliverable D5.2.

Creation of a new SCGT application for the creation of pages was found to be time consuming, largely due to incomplete instructions for use. The user was required to abstract what had to be done from the two demonstration examples. A manual or an SCGT creation Tool would have been more appropriate. However, Viataal found the creation of their interface, described in XML, a relatively easy task for anyone with reasonable technical skills.

Once the application had been developed, the advantages of the SCGT were well proven by Viataal's experience. Users who wished to publicise a job vacancy were able to make a summary of a vacancy description in NGT as a web page in a couple of minutes.

4 A Survey of Existing CMS and Web Development Processes

4.1 Introduction

Content developed using eSIGN tools will become out of date quickly unless working practices keep signed content updated in line with changes to text pages. It was anticipated that maintainers of these websites would already be using content management systems. It was considered important to understand current practice of eSIGN partners in order to develop processes that would make it straightforward to keep signed content up to date.

4.2 Content Management for the German Portal

Systematics Integrations use a number of content management systems.

Vignette (http://www.vignette.com) provides a core Content Management application which can be linked to business processes. The Process Workflow Modeler provides a GUI interface for defining workflow that can be linked to the content management application.

CoreMedia (http://www.coremedia.com/) Smart Content Technology enables content to target a number of output platforms: Web, PDAs, WAP, etc. It has multilingual support and addresses WAI guidelines. A range of editing clients is available, principally web-based, using WYSIWYG editing. It is also possible to import content in a number of formats including XML. Tools exist for defining flexible workflows, with a workflow modeller.

4.3 Content Management on British Sites

The example BSL sites are developed without using content management systems, although pages are delivered using technologies such as ASP.

It had been anticipated that the Norfolk portal would be well established when the eSIGN project started, but a change of contractors meant that it was not possible to integrate signed pages into the main Norfolk website.

4.4 The Viataal approach

Viataal's website is being created and maintained through a Content Management System developed to meet their needs. Content for the pages is stored in a database and the pages are built dynamically when a visitor requests a particular page.

Currently the CMS enables the user to build webpages with text, images, download links, and hyperlinks. The CMS-tool provides a simple interface to upload image files or the files offered for download.

5 Content Styles

5.1 Introduction

All web pages including eSIGN content will need to include scripting to launch the avatar or link to a window or pane already containing the avatar. Scripts will send SiGML data to the avatar at the appropriate time, often in response to requests from users via buttons or hyperlinks.

The page design must allow for the avatar to be embedded or for the text and image content to be visible alongside the avatar. This may require some reorganisation and may require content with signing to be presented over several pages where a single page sufficed for plain content. The issues are similar to those of presenting

information for different target devices such as PDAs and WAP phones. Since advanced CMSs already address these issues, accommodating signing presents relatively few new issues.

Once the design has been decided, maintaining content is fairly straightforward since SiGML data is in XML format. Hence it can be treated as marked-up text by a CMS as for (X)HTML. Scripts can also be written so that the SiGML content is in a separate file accessed via a URL.

5.2 Free form text

To modify standard text on web pages for signed translation, the main requirement is to add some form of link that will trigger signing for a particular section. eSIGN partners have used both small icons, and hyperlinking the text itself as solutions.

Sections should be relatively small; signing a complete page of continuous text as a single block would be inappropriate as it would not allow deaf users the same freedom to read short sections and skip sections, in the same way that other readers of the page might do.

Sign language translators also need to consider if an exact translation is most useful, or whether shorter summaries, or more expansive descriptions would be more helpful. This would differ according to the context.

Content will generally be prepared using the eSIGN Editor and entered into static pages or a CMS that will generate pages dynamically.

5.3 Forms

Most web pages which have form-filling content are either self-explanatory, or have some guidance to aid completion. For deaf people, a signed translation of such guidance may be sufficient, or additional explanation may be necessary. Advice from deaf people and other sign language experts on such matters is encouraged. These matters aside though, content style for forms is essentially the same as that used for free form text.

Since the additional signing to be provided with forms is in free form, the eSIGN Editor will be the appropriate tool for generating SiGML sequences.

5.4 Structured content

Creating structured content is a familiar application of a CMS. Pages are dynamically generated based on database contents so structured content requires that the database is populated in an appropriate fashion.

Applications built using the SCGT generally use a template that produces a complete web page. For use with a CMS it will be more appropriate to generate information that can be transferred to the CMS database.

There is no reason why content created using the SCGT could not be extended to create output pages for hearing people as well as sign language versions. The CMS could use the information to generate pages in any number of languages on demand.

An SCGT application uses signing phrases and text that make up a complete model of the domain to be covered by the application. The signed phrases will be prepared using the eSIGN Editor but generation of the final content does not require the user to have any knowledge of signing.

6 Content Creation for Deaf Users

6.1 Introduction

Since signed material produced using eSIGN tools appears as conventional XML text in SiGML, many aspects of content creation are the same whether the output is for hearing or deaf people. However, special techniques are required to design sites that present content adapted to the needs of deaf people and to accommodate the avatar used to display signing.

6.2 Handling multilingual content

Translation of web page content for deaf users raises many of the same issues as translation for any other *spoken* language. A translation must be prepared, and then applied to existing page structures in some way. When a page is edited, the maintenance process ensures that it triggers the re-translation of corresponding pages in other languages.

6.3 The particular challenges of sign language content

Obviously the major difference to web page modification where a sign language translation is required is the inclusion of an avatar somewhere on the page. The avatar is relatively slow to load, and consequently, once present, it is desirable that it stays loaded to sign further text as required. This meant that all three partners adopted some version of a Frames format to allow it to remain while a user navigated though the pages of a website. Existing pages had to be modified accordingly. Two principal solutions were adopted by eSIGN partners:

- The avatar is included as a frame, taking up to a third of the space previously occupied by a web page.
- The avatar is displayed in a small floating modeless dialog box (itself in fact a frame) which can be moved around by the user to facilitate viewing of web pages beneath.

Instructions on how to create both versions are given as part of the Web Page Cookbook in the Appendix.

If the chosen format is to use the floating modeless dialog box, then modification to the existing web page will be minimal, requiring only the addition of small icons or hyperlinks to trigger signing. However, adding the avatar to the web page itself will require considerable restructuring of the original page. Where there are a large number of graphics or photographs on a page, the difficulties could even make alteration infeasible.

To cater for various screen resolutions, many web designers now choose to build pages to fill the screen at a resolution of 800x600 pixels. Viewed at higher resolutions this results in a considerable amount of blank space around the page content. Taking advantage of this space, leaving the existing page as it was and encouraging viewing at a higher resolution, could be one solution where restructuring of pages is a problem.

6.4 Linking the Editor to Content Management Systems

Displaying signed content should present no difficulties for a CMS. As emphasised above, SiGML is just marked-up text and can therefore be stored in the CMS database in the same way as other XML-based data such as XHTML. The templates and stylesheets used for displaying signed text will be more sophisticated as they

must include scripting for loading and interacting with the avatar. Nevertheless, the principles are no different from those needed for rich multimedia content.

It may be more difficult to link the CMS to the tools used to produce signed content (the eSIGN Editor and SCGT applications). In particular there will be special skills needed to use the eSIGN editor and it may be usual for specialist bureaux to generate such content. Hence the workflow for a CMS may need to support the export and import of signed documents that are edited off site.

7 Portal Integration in eSIGN

7.1 Introduction

When eSIGN was planned, it was envisaged that those managing eGovernment portals would use sophisticated content management systems to deliver information to clients. Since the project would focus on creating example content using specially developed tools, there would be a danger that eSIGN content would not be updated along with content for hearing users and would rapidly become obsolete.

The need was identified to consider links between eSIGN tools and portal software in order to provide eGovernment content creators with a practical means of including signed content along with multilingual content and content for users with special needs. In the event, the portals used by those deploying eSIGN content are less sophisticated than was envisaged so the need for linking is only now becoming important.

7.2 Integration with the German Portal

Although most pages generated for hamburg.de have static content, it is becoming apparent that it is too time consuming to transfer and incorporate SiGML data by hand as pages are updated.

A good deal of the information that is involved in a web page is used by the eSIGN Editor when preparing signed content and is present in the XML-based document format that includes the original translated text as well as the corresponding SiGML. Rather than exporting individual SiGML fragments, the approach will be to use XSLT (XML stylesheet transformations) to combine the information from the Editor with other data to be presented on signed web pages.

7.3 Integration with the British Portal

The site for BSL eSIGN content in Norfolk is maintained separately from the main Norfolk portal at norfolk.gov.uk. Since relatively few types of pages are maintained, it is appropriate to use manual editing through commodity web design software. By placing SiGML content in separate files, accessed through URLs, revised content can be installed quickly.

Other eSIGN content in BSL uses the VANESSA system which is a freestanding application rather than a website. The rate of change of content presented by a VANESSA system can be handled by manual editing supported by special tools as described earlier.

7.4 Integration with the Dutch Portal

Since Viataal's website is being created and maintained through a bespoke CMS it would be possible to integrate handling of signed data in SiGML with the existing database.

The primary application that has been implemented uses the SCGT to generate information about job vacancies for deaf people. In its current form the application enables individual pages to be created and stored in HTML files. While the number of pages in use at a given time is moderate, it is practical to create a manual index.

If the numbers of pages became substantial it would be necessary to automate the process of publishing and removing vacancies, as well as maintaining indexes of live vacancies. Rather than creating HTML pages directly, a more flexible approach would be to generate an XML file that could be processed (using XSLT for example) to populate a database or to produce both HTML pages and index information.

Appendix – The Web Page Cookbook

The Web Page Cookbook is an instruction manual to help web page developers add signing functionality to web pages. Starting with the most basic page, developers can then select from a variety of additional features to enhance the look and use of their pages. For each feature, details of the HTML code and/or scripting required, and where they should be placed in the HTML are given, together with screenshots to show the results as appropriate.

A reasonable degree of familiarity with HTML and Javascript is assumed.

Developers should ensure that the SiGMLSigning avatar is installed, and that HTML pages are viewed in Internet Explorer v6

A reference section at the end gives a list of most of the SiGMLSigning variables and functions available for scripting.

CONTENTS

1 WEBPAGES 2

1.1 The Basic Webpage 2

1.2 More basic facilities 4

1.3 Framesets 11

2 ADDITIONAL FEATURES 19

2.1 – How to check SiGMLSigning software is installed on a client machine 19

2.2- How to change the avatar background colour 20

2.3 – How to switch ambient motions on and off 20

2.4 – How to use alternative directories for SiGML files 20

2.5 – How to avoid putting all the SiGML in the HTML pages 21

3 - REFERENCE 22

3.1 – Available Avatars 22

3.2 - SiGMLSigning - Enumerations, Methods, Properties And Events 23

1 WEBPAGES

1.1 The Basic Webpage

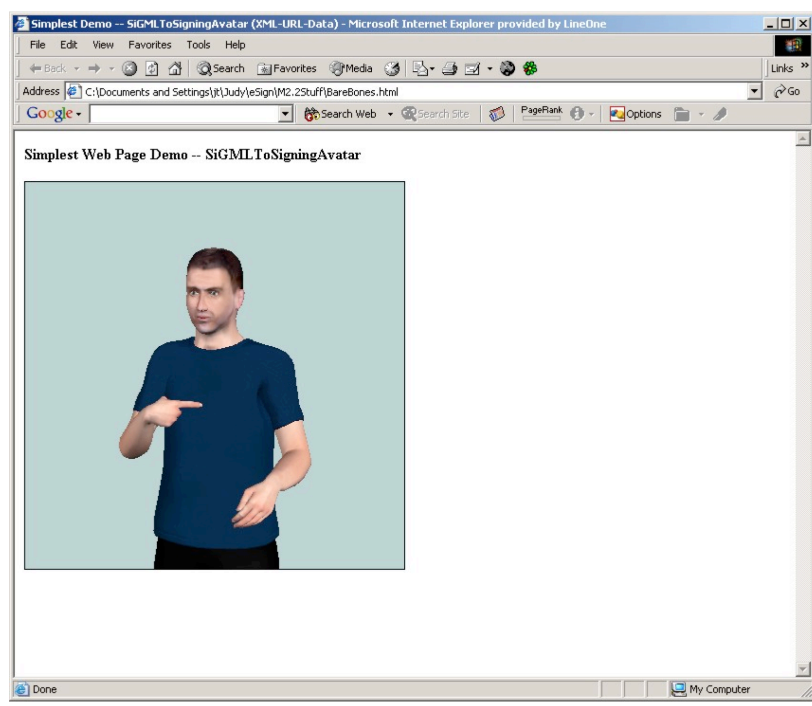

Figure 1
The basic webpage with avatar , created from the code on the following page

Notes on the HTML code used to create the Simplest Web Page Demo:

- Javascript method names are not fixed, though obviously the SiGMLSigning methods and events used are.
- Alternative avatars are available. VGuido is the standard eSIGN avatar (`var avatarID = 6` as shown in Note 2 on the previous page) A list of alternative avatars is given in Section 3.1.
- The `AcceptNewSiGMLString` method creates animation frames from the prescribed SiGML, but the animation is not played until the `PlayerChangeMode(1)` method is called. The '1' parameter is a boolean, setting the mode to 'Auto'. When the signing is complete, the mode automatically reverts to mode '0' or 'Manual', and the `OnPlayerChangeMode` event is triggered.
- A Complete list of Methods, Properties and Events for SiGMLSigning is given as Section 3.6.2

As a starting point, the following code will produce a basic HTML page, containing an avatar that signs one simple sign. Notice the `<HEAD>` section contains a number of scripts, and the `<BODY>` section has the avatar object definition. The numbered labels describe page events in order.

Code for Simplest Demo

```
<HTML>
	<HEAD>
		<TITLE>Simplest Demo -- SiGMLToSigningAvatar (XML-URL-Data)</TITLE>

		<SCRIPT LANGUAGE="javascript">
			var jSiGMLIn;

			function Initialise()
			{
				var avatarID = 6;
				jSiGMLIn = new ActiveXObject("JSiGMLIn.JSiGMLInput");
				ssAvatar.Initialise(avatarID, jSiGMLIn);
			}

			function doAvatarReady()
			{
				var sigml = "<sigml> <signing_ref uri='iTakeMugG' format='SiGML'/> </sigml>";
				var fps = 25;
				ssAvatar.AcceptNewSiGMLString(sigml, fps);
			}

			function doStartAnimation()
			{
				ssAvatar.PlayerChangeMode(1);
			}

		</SCRIPT>

		<SCRIPT for="ssAvatar" event="OnAvatarInitialised()" language="javascript">
			doAvatarReady();
		</SCRIPT>

		<SCRIPT FOR="ssAvatar" EVENT="OnSigningDataIsAvailable()" LANGUAGE="javascript">
			doStartAnimation();
		</SCRIPT>
	</HEAD>

	<BODY ONLOAD="Initialise()" >
		<H3>Simplest Web Page Demo -- SiGMLToSigningAvatar</H3>
	<BODY ONLOAD="Initialise()" >
		<H3>Simplest Web Page Demo -- SiGMLToSigningAvatar</H3>

		<!-- SiGMLToSigningAvatar -->
		<OBJECT ID="ssAvatar"
			CLASSID="clsid:20E52958-E0E3-4C06-8D0E-A477DEFD4933"
			STYLE="height: 75%; width: 50%;">
		</OBJECT>
	</BODY>
</HTML>
```

2. Javascript `Initialise()` function.
Sets avatarID to 6 = Virtual Guido.
Creates `JSiGMLIn` object.
Calls the avatar object's "Initialise" method to set avatar to be used and passes pointer to JSiGMLIn library. Triggers `OnAvatarInitialised` event when complete

4. `doAvatarReady()` function.
Define sigml to be used (there are other ways of doing this)
Also define frame speed. Pass both to avatar method `AcceptNewSiGMLString` to create animation frames.

6. `doStartAnimation()` function
`ssAvatar.PlayerChangeMode(1)` changes the mode to Auto.
This causes the avatar to start playing the created animation.

3. `OnAvatarInitialised` event triggered when `ssAvatar.Initialise(avatarID.jSiGMLIn)` has been completed. Calls `doAvatarReady()` function

5. `OnSigningDataIsAvailable` event triggered when `AcceptNewSiGMLString` method has finished creating animation frames for SiGML specified. Calls `doStartAnimation()` function

1. When page is loaded, call javascript "Initialise()" function

Page heading

Avatar object definition.
height and width %'s may be altered

1.2 More basic facilities

This is a reworking of the *Form-Driven Transaction Support Toolkit* supplied as Milestone M2.2 of the eSIGN project. It shows how to present a play-list of SiGML documents, with a synchronized display of text accompanying each SiGML sequence in the play-list. This is achieved using the `SiGMLToSigningAvatar ActiveX` Control.. The sequence is repeated unless the 'Paused' checkbox is ticked. The resultant web page is shown below as Figure 2.

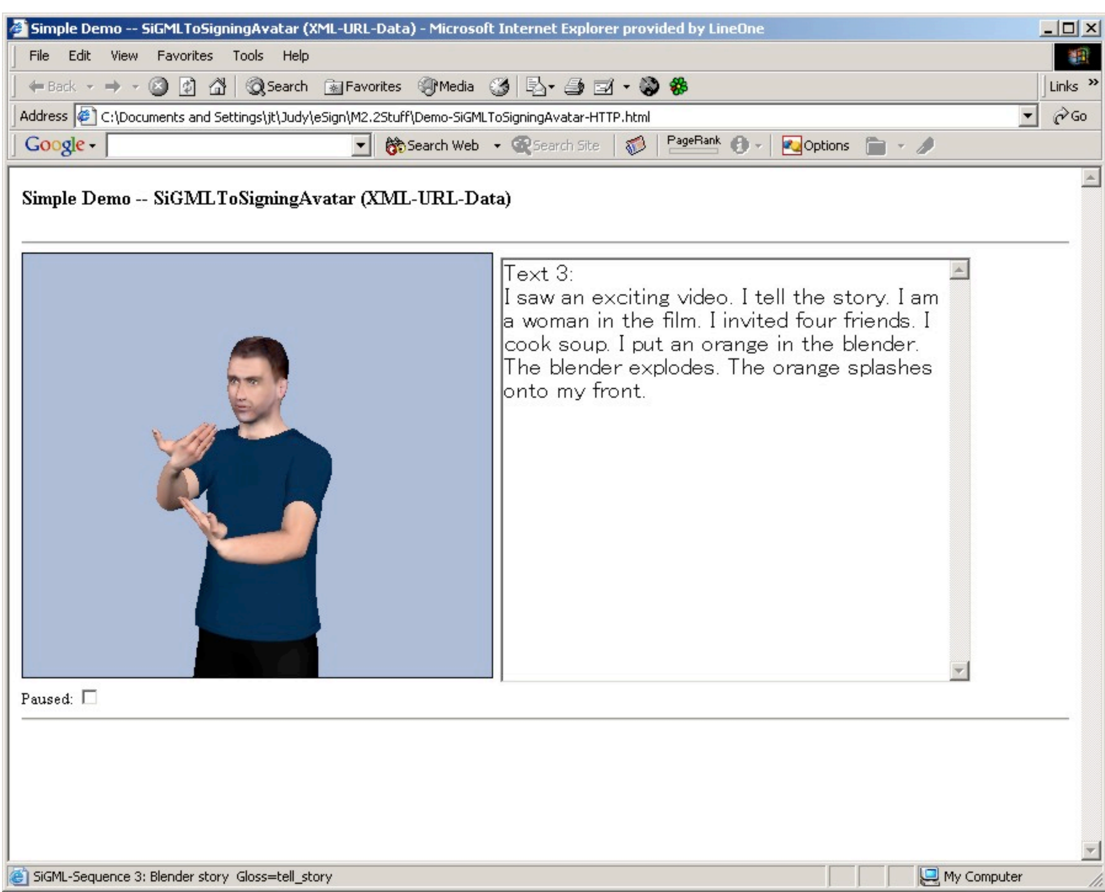

Figure 2 – Basic signing web page as used in Milestone M2.2

Notes about this page, together with the code, are given below.

Each *SiGML* document in the play-list is identified by an XML "data island": here the data island contains a relative URL identifying the required *SiGML* document, but it could equally well contain the text of the *SiGML* sequence itself in-line (or the path name on the local file system of a file containing the *SiGML* document).

Javascript code is used to control the dynamic animation. When the page is loaded its `Initialise()` function is invoked. This populates the arrays which hold the data associated with each *SiGML* document in the play-list and initialises the cyclic play-list index. It then creates an instance of the `JSiGMLInLib` control (which provides *SiGML* input and translation) which allows the avatar's background colour to be set, and allows the playing speed of each animation to be varied. Finally, once an instance of a `ISiGMLToSigningAvatar` has been instantiated the `ssAvatar.Initialise(avatarID, jSiGMLIn)` is called to specify the avatar to use and to pass in a pointer to a `JSiGMLIn` library.

There are Javascript handlers for the three main SiGMLToSigningAvatar control events: `OnAvatarInitialised()`, `OnSigningDataIsAvailable()`, `OnPlayerChangeMode()`.

- The `OnAvatarInitialised() event signifies that a`n avatar has been successfully loaded. This event is always fired after a call to `Initialise()` or `SwitchAvatar()`. The script for the event calls the `doAvatarReady()` method, which in turn calls `startNextSiGMLText()`. This displays the text and then the line `ssAvatar.AcceptNewSiGMLString(sigml, fps)` which passes in a SiGML string for which animation frames are required.
- Once the frames have been generated the `OnsigningDataIsAvailable()` event is fired, but the player does not automatically start playing the newly generated animation frames The handler for this event does this explicitly by invoking the player's `PlayerChangeMode()` method with "auto" as the `mode` parameter value.
  The `OnSigningDataIsAvailable` event calls `doStartAnimation()` which calls `ssAvatar.PlayerChangeMode(pmAuto)` as described.
- The next event to occur is `OnPlayerChangeMode()` which will happen when the avatar has finished signing. The event has a flag parameter indicating the new mode (`auto` or `manual`); hence the handler for this event needs to check that the change is from `auto` to `manual` before taking the actions appropriate to the end of a play-list step.
  The event calls the `doPlayerModeHasChanged(bManual)` method which advances the cyclic play-list index and initiates a (1000ms) timer. The handler for the this timer event normally causes the initiation of the next step in the play-list cycle, but if the page's `Paused` check-box is set, then another (1000ms) timer is initiated: this has the same handler, so the next step in the play-list cycle will only be initiated when the `Paused` check-box is cleared.

Throughout the playing sequence, progress information is presented in the browser's status bar.

The handler for the `AvatarReady()` event causes the next *SiGML* sequence to be fed to the player via its `PlaySiGML()` method, together with the display of the corresponding text in the page's text area.
In due course the `DoneGeneration()` event occurs, indicating that the player has now generated the animation frames for this *SiGML* sequence: its handler causes the page's `playing` flag to be set, while the player proceeds automatically to play the newly generated frames.
When the playing of these animation frames is completed the player generates the `DonePlay()` event, whose handler advances the cyclic play-list index and initiates a (1000ms) timer.
The `SiGMLToSigningAvatar` control generates an event as each frame is played, and it also generates an event as the animation of each new *SiGML* sign is started. By providing handlers for these events, this page is able to provide information in the browser's status bar. Specifically, the gloss name of the currently playing sign within a

given *SiGML* sequence is displayed while that sign plays, and at the end of each *SiGML* document the status display briefly records the number of animation frames shown, and the total number for that sequence: this gives an indication of the graphics capabilities of the given computer platform.
Two further events, `OnPlayerFinishShowFrame(lFrameIndex,lFinishTimeMillis)` and `OnPlayerChangeUnit(lIndex)` increment the frameCount, and displays the new gloss of the sign being played in the browser's status bar.

[Note it is possible to use a `<signing_ref>` *SiGML* element to identify each *SiGML* document file to the `SiGMLToSigningAvatar` control. However, for the current release of the *SiGMLSinging* package, this technique works only for files identified by a local file system pathname and not for arbitrary URLs.]

Complete code listing:

```
<HTML>
	<HEAD>
		<TITLE>Simple Demo -- SiGMLToSigningAvatar (XML-URL-Data)</TITLE>

		<XML ID="SiGMLDoc0" SRC="./iTakeMugDemo.sigml"/>	//XML Data islands
		<XML ID="SiGMLDoc1" SRC="./forkTwiceDemo.sigml"/>
		<XML ID="SiGMLDoc2" SRC="./werkNGTDemo.sigml"/>
		<XML ID="SiGMLDoc3" SRC="./blenderStoryDemo.sigml"/>

		<SCRIPT LANGUAGE="javascript">
			// Definitions from SiGMLToSigningAvatar control:
			var pmManual = 0;
			var pmAuto   = 1;

			var atVisia3 = 1;
			var atVisia2 = 2;
			var atVisia4 = 3;
			var atARPMan = 4;
			var atNewVisia3 = 5;
			var atVGuido = 6;

			var playLabels;     // Tags for SiGML sequences;
			var playTexts;      // NL Texts for SiGML sequences;
			var playLimit;      // Number of SiGML sequences;

			var curPlayIndex;   // Current SiGML sequence;
			var curSiGMLLabel;
			var curText;
			var curStatusMsg;

			var jSiGMLIn;

			var animSpeedUp = 1.0;      // SiGMLToSigningAvatar properties;
			var fps = 25;

			var avatarID = atVGuido;    // The chosen avatar;
			var playing = false;

			var frameCount = 0;         // Counts frame-shown events.

			function Initialise()
			{
			    playLimit = 4;
			    playLabels = new Array(
			        "I-Take-Mug", "Fork-twice", "Werk-NGT", "Blender story"
			    );
			    playTexts = new Array(
			        "Text 0: \nI take the mug.",
			        "Text 1: \nFork, fork.",
			        "Text 2: \nWerk.",
```

```
            "Text 3: \n"+
                "I saw an exciting video. I tell the story. "+
                "I am a woman in the film. I invited four friends. "+
                "I cook soup. I put an orange in the blender. "+
                "The blender explodes. The orange splashes onto my front.\n"
        );
        curPlayIndex = 0;

        jSiGMLIn = new ActiveXObject("JSiGMLIn.JSiGMLInput");
        ssAvatar.BackgroundColour = 0xD0B0A0;   // BGR (not RGB!)
        ssAvatar.Initialise(avatarID, jSiGMLIn);
        window.status = "Avatar ("+avatarID+") loading.";
    }

    function Terminate()
    {
        ssAvatar.Terminate();
        jSiGMLIn = null;
    }

    function getSiGMLText(index)
    {
        var sdoc = document.all("SiGMLDoc"+index);
        var sigml = sdoc.documentElement.xml;

        return sigml;
    }

    function doAvatarReady()
    {
        window.status = "Avatar ("+avatarID+") loaded.";
        startNextSiGMLText();
    }

    function startNextSiGMLText()
    {
        var i = curPlayIndex;

        curSiGMLLabel = playLabels[i];
        curText = playTexts[i];
        curStatusMsg = "SiGML-Sequence "+i+": "+curSiGMLLabel;

        var sigml = getSiGMLText(i);
        window.status = curStatusMsg;
        txtShowText.value = curText;

        ssAvatar.PlayerSpeedLog2 = animSpeedUp;
        ssAvatar.AcceptNewSiGMLString(sigml, fps);
    }

    function doStartAnimation()
    {
        frameCount = 0;
```

```
                playing = true;
                ssAvatar.PlayerChangeMode(pmAuto);
            }

            function doPlayerFrameShown()
            {
                ++ frameCount;
            }

            function doPlayerSiGMLUnitChange(index)
            {
                var glossMsg = "  Gloss="+ssAvatar.GetSiGMLUnitName(index);
                window.status = curStatusMsg+glossMsg;
            }

            function doPlayerModeHasChanged(bManual)
            {
                if (bManual == 1)
                {
                    var fmsg = ""+frameCount+" frames out of "+ssAvatar.CountFrames;
                    window.status = curSiGMLLabel+" animation done ("+fmsg+").";
                    ++ curPlayIndex;
                    if (curPlayIndex == playLimit) {
                        curPlayIndex = 0;
                    }
                    playing = false;
                    startTicking();
                }
            }

            function tick()
            {
                if (!playing)       // Sanity check -- should never fail;
                {
                    if (chkPaused.checked) {
                        startTicking();
                    }
                    else {
                        startNextSiGMLText();
                    }
                }
            }

            function startTicking()
            {
                var timerID = window.setTimeout("tick()", 1000);
            }
        </SCRIPT>

        <SCRIPT for="ssAvatar" event="OnAvatarInitialised()" language="javascript">
            doAvatarReady();
        </SCRIPT>
```

```
		<SCRIPT FOR="ssAvatar" EVENT="OnSigningDataIsAvailable()" LANGUAGE="javascript">
			doStartAnimation();
		</SCRIPT>

		<SCRIPT FOR="ssAvatar" EVENT="OnPlayerChangeMode(bManual)" LANGUAGE="javascript">
			doPlayerModeHasChanged(bManual);
		</SCRIPT>

		<SCRIPT FOR="ssAvatar" EVENT="OnPlayerFinishShowFrame(lFrameIndex, lFinishTimeMillis)"
		LANGUAGE="javascript">
			doPlayerFrameShown();
		</SCRIPT>

		<SCRIPT FOR="ssAvatar" EVENT="OnPlayerChangeUnit(lIndex)" LANGUAGE="javascript">
			doPlayerSiGMLUnitChange(lIndex);
		</SCRIPT>
	</HEAD>
```

```
	<BODY ONLOAD="Initialise()" ONUNLOAD="Terminate()">

		<h3>Simple Demo -- SiGMLToSigningAvatar (XML-URL-Data)</h3>

		<!-- SiGMLToSigningAvatar -->
		<OBJECT ID="ssAvatar"
		    CLASSID="clsid:20E52958-E0E3-4C06-8D0E-A477DEFD4933"
		    STYLE="height: 65%; width: 45%;">
		Unable to display SiGMLToSigningAvatar Control
		</object>
		 
		<TEXTAREA ID="txtShowText"
		    STYLE="height: 65%; width: 45%; font: '14pt sans-serif';">
		</TEXTAREA>
		<br>
		Paused:
		<input TYPE="CHECKBOX" NAME="chkPaused" VALUE="1"></input>
	</BODY>
</HTML>
```

1.3 Framesets

The eSIGN avatar is relatively slow to load, and therefore, once present, it is desirable that it stays loaded to sign further text as a user navigates though the pages of a website. Existing pages need to be modified. Two basic solutions are offered here:

- The avatar is included as a frame, taking up some of the space previously occupied by the web page it is to translate into sign language.
- The avatar is displayed in a small floating pop-up window, or modeless dialog box (itself in fact a Frame) which can be moved around by the user to facilitate viewing of web pages beneath.

For the purposes of comparison, both versions start from the same basic page(s):

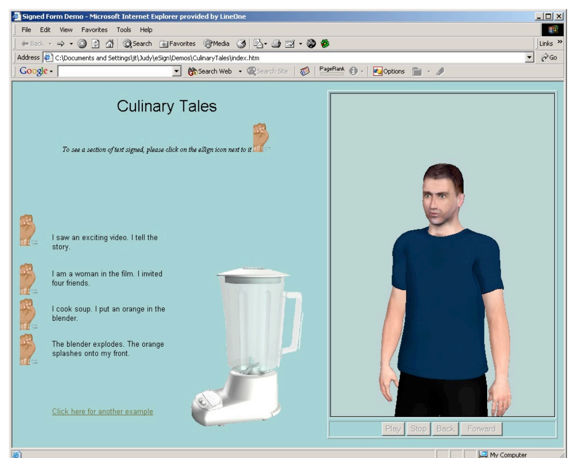

Figure 3a: Avatar included as a Frame. Original page (see Figure 3c) has to be modified to include avatar.

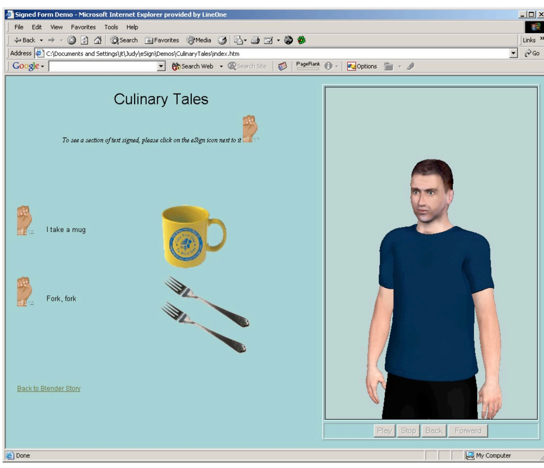

Figure 3b: Avatar remains while a new linked page is loaded.

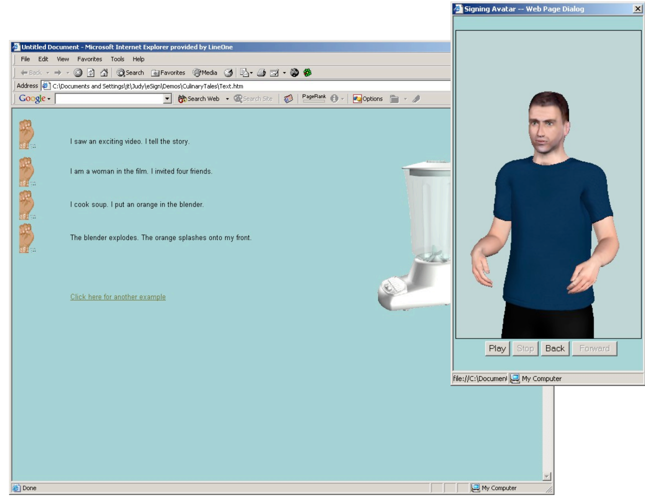

Figure 3c: Avatar included as a Pop-up window which remains even when linked pages below change. Changes to original page are minimal.

In both versions, small icons have been inserted next to the text which is to be signed. Hyperlinks on the text could be used instead. Javascript is used to link the icons to the Avatar.play method, which instructs the avatar to play the appropriate animation.

For each version of the solutions, there follows a diagram showing the main sections of code which initiate avatar signing, and how this is linked in the frameset. The complete code for the page is also included for the first solution. The minor modifications necessary to alter that code to make a pop-up version should be clear from the second diagram.

Avatar integral with the web pages In this version, the avatar is loaded at the same time as the first frameset webpage. Any number of frames can be used to make up a single web page, which will include one frame for the avatar. It is advisable to use only a few frames. Linked pages do not need to include the avatar, as they will appear as a single frame constituent, to replace one frame of the first webpage. You need a directory index page (usually index.html) giving details of all the html pages used to build the frameset and their layout, and all the other html pages you plan to use for the first frameset, including one for the avatar.

Index.htm (frameset, or top) There is no reason why you cannot change this layout (no title, title across full pagewidth, etc)

```
<head>
<title>Signed Form Demo</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<frameset rows="*" cols="*,421" framespacing="0" frameborder="NO" border="0">
  <frameset rows="135,*" cols="*" framespacing="0" frameborder="NO" border="0">
    <frame src="Title.htm" name="topFrame" scrolling="NO" noresize>
    <frame src="Text.htm" name="mainFrame">
  </frameset>
  <frameset rows=370">
    <frame src="Avatar.htm" name="rightFrame" scrolling="NO" noresize >
</frameset>
<noframes>
      <body></body>
</noframes>
```

A frame*set* with 1 row and 2 columns
The first column has 2 rows called `Title.htm` and `Text1.htm`

The second column has just one row called `Avatar.htm`

(the last 3 lines are for browsers which do not support frames)
NOTE the <frameset> tags and content are OUTSIDE the <HEAD></HEAD> and <BODY> </BODY> tags

Title.htm (topFrame)

(Probably won't have any linking material)

Text.html (mainFrame)

All the first regular page plus image or text links to signing, in the format:

```
<a
ref="javascript:window.top.rightFrame.LoadPlayList('a')">
          <img src="images/Esign_logo03_micro.jpg"
          width="31" height="57" border="0">
 </a>
```

(this could all be on one line)

Notice that in `window.top.rightFrame.LoadPlayList('a')`…
`window.top` refers to the `index.htm` frameset which in turn tells you that the javascript function `Play()` is in the `rightFrame` of that frameset.
`'a'` is a way of referencing the XML that should be played by the avatar for this particular link. Any letter(s) or numbers could be used instead of `'a'`.

Any linked pages will replace this frame, while `Avatar.htm` and `Title.htm` remain.

Avatar.htm (rightFrame)

<HEAD> … </HEAD> containing:
All javascript functions to control the avatar, buttons etc, including LoadPlayList (**'a'**)

(NB In the Initialise() function, order is important:

```
ssAvatar.BackgroundColour = 0xD0B0A0;
ssAvatar.Initialise(avatarID, jSiGMLIn);
ssAvatar.UserInputBaseURI = C:\MyDir;
```
)

Also include the XML
(or references to it – syntax described elsewhere in this document), eg:

```
<XML id="a">
   <sigml>
       <hamgestural_sign gloss="video">   … etc.
```

note the `"a"` here corresponds to the `'a'` in `Play('a')` in `Text.html`

```
<BODY bgcolor="#CCCCFF" ONLOAD="Initialise()">
```

…is the start of the <BODY> setting the background colour and calling the `Avatar.Initialise()` function.
The <BODY> </BODY> tags also contain the Avatar ActiveX object definition:

```
<object classid="clsid: 20E52958-E0E3-4C06-8D0E-A477DEFD4933"
  width = "100%;"height = "100%;" align="right" id="Avatar"  >
</object>
```

Also include any buttons, slider bars etc which control Avatar play.

Code for the Frameset with integral Avatar

There are four separate files which go to make up this page: Index.htm, Title.htm, Text.htm and Avatar.htm. A Second frame Text2.htm, linked from Text.htm is also included. Text2.htm replaces Text.htm in the frameset when the “Click here for another example” link is clicked.

Index.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Culinary Tales, Integral Avatar version, Index.htm</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<frameset rows="*" cols="*,421" framespacing="0" frameborder="NO" border="0">
  <frameset rows="210,*" cols="*" framespacing="0" frameborder="NO" border="0">
    <frame src="Title.htm" name="topFrame" scrolling="NO">
    <frame src="Text.htm" name="mainFrame">
  </frameset>
    <frame src="Avatar.htm" name="rightFrame" scrolling="NO" noresize >
</frameset>
<noframes><body>

</body></noframes>
</html>
```

Title.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Culinary Tales, Integral Avatar version, Title.htm</</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#99CCCC">
<table width="100%" border="0" align="center" cellspacing="10">
  <!--DWLayoutTable-->
  <tr>
    <td valign="top" bgcolor="#99CCCC"><div align="center"><font size="6" face="Geneva, Arial, Helvetica, sans-serif">Culinary
        Tales</font> </div></td>
  </tr>
  <tr valign="middle">
    <td valign="top" bgcolor="#99CCCC"><div align="center"><font size="2" face="Times New Roman, Times, serif"><em>To
        see a section of text signed, please click on the eSign icon next to it</em><strong>
                  </strong></font><font face="Verdana, Arial, Helvetica, sans-serif"><img src="images/Esign_logo03_micro.jpg" width="31"
align="bottom"></font></div></td>
  </tr>
</table>
</body>
</html>
```

Text.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Culinary Tales, Integral Avatar version, Text1.htm</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#99CCCC">
<table  border="0" align="CENTER" width="100%">

  <tr>
    <td width="32" align="left" valign="top"><div align="left"><font face="Verdana, Arial, Helvetica, sans-serif"><a 
href="javascript:window.top.rightFrame.LoadPlayList('a')"><img src="images/Esign_logo03_micro.jpg" width="31" height="57" 
border="0"></a></font></div></td>
    <td width="30" valign="middle"><!--DWLayoutEmptyCell--> </td>
    <td width="372" valign="middle"> <p> 
      <p><font size="3" face="Geneva, Arial, Helvetica, sans-serif">I saw an exciting 
        video. I tell the story.</font><br>
        <br>
    </td>
    <td width="250" rowspan="6" align="center" valign="bottom"><div align="right"><img src="images/blender-right-front%20copy.gif" width="198" 
height="279"></div></td>
  </tr>
  <tr>
     td  valign="top"><font  face="Verdana,  Arial,  Helvetica,  sans-serif"><a  href="javascript:window.top.rightFrame.LoadPlayList('b')"><img 
src="images/Esign_logo03_micro.jpg" width="31" height="57" border="0"></a></font></td>
    <td valign="middle"><!--DWLayoutEmptyCell--> </td>
    <td valign="middle"><p><font size="3" face="Geneva, Arial, Helvetica, sans-serif">I 
        am a woman in the film. I invited four friends.</font></p></td>
  </tr>
  <tr>
    <td align="left" valign="top"><div align="left"><font face="Verdana, Arial, Helvetica, sans-serif">
<a href="javascript:window.top.rightFrame.LoadPlayList('c')"><img src="images/Esign_logo03_micro.jpg" width="31" height="57" 
border="0"></a></font></div></td>
    <td valign="middle"><!--DWLayoutEmptyCell--> </td>
    <td valign="middle"> <p><font size="3" face="Geneva, Arial, Helvetica, sans-serif">I 
        cook soup. I put an orange in the blender.</font><br>
      </p></td>
  </tr>
  <tr>
    <td valign="top"><font face="Verdana, Arial, Helvetica, sans-serif"><a href="javascript:window.top.rightFrame.LoadPlayList('d')"><img 
src="images/Esign_logo03_micro.jpg" width="31" height="57" border="0"></a></font></td>
    <td valign="middle"> </td>
    <td valign="middle"><p><font size="3" face="Geneva, Arial, Helvetica, sans-serif">The 
        blender explodes. The orange splashes onto my front.</font><br>
      </p></td>
  </tr>
  <tr>
    <td height="3" valign="top"><font face="Verdana, Arial, Helvetica, sans-serif"> </font></td>
    <td></td>
    <td><p> </p>
```

```
      <p> </p>
      <p><a href="Test.htm"><font face="Geneva, Arial, Helvetica, sans-serif">Click
        here for another example</font></a></p></td>
  </tr>
  <tr>
    <td height="3" valign="top"> </td>
    <td></td>
    <td></td>
  </tr>
</table>
</body>
</html>
```

Avatar.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Untitled Document</title>
<SCRIPT LANGUAGE="javascript">
               var H2S;
               var sGlossBeingPlayed = new String;
               var signSet;

               function Initialise()
               {
                       H2S = new ActiveXObject( "JSiGMLIn.JSiGMLInput");
                       SetButtonsStatus();
                       Avatar.Initialise( 6, H2S);   // 0=NoAvatar, 1=Visia3, 2=Visia2, 3=Visia4, 4=ARPMan, 5=NewVisia3, 6=Guido
                       Avatar.DoBlendFrames = true;  // 8=Michael
               }

               function LoadPlayList(signSet)
               {
                       var docXML;
                       //alert(signSet);
                       docXML = document.all(signSet).XMLDocument;
                       docXML.async = false;
                       //window.alert( docXML.xml);
                       Avatar.AcceptNewSiGMLString( docXML.xml, 25);
               }

               function Play()
               {
                       //Avatar.PlayerDoCycleFrames = chkCycle.checked;
                       //LoadPlayList(signSet);
                       Avatar.PlayerDoCycleFrames = false;
                       Avatar.PlayerChangeMode(1);
                //SetButtonsStatus();
               }

               function Stop()
               {
```

```
        Avatar.PlayerChangeMode( 0);
        SetButtonsStatus();
}

function Back()
{
        Avatar.PlayerShowPreviousFrame();
        SetButtonsStatus();
}

function Forward()
{
        Avatar.PlayerShowNextFrame();
        SetButtonsStatus();
}

function SetButtonsStatus()
{
        btnPlay.disabled = !(Avatar.SigningDataAvailable && Avatar.PlayerModeIsManual);
        btnStop.disabled = Avatar.PlayerModeIsManual;
        btnBack.disabled = !Avatar.PlayerShowPreviousFrameOK;
        btnForward.disabled = !Avatar.PlayerShowNextFrameOK;
}

function CycleClicked()
{
        Avatar.PlayerDoCycleFrames = chkCycle.unchecked;
}

</SCRIPT>

<SCRIPT FOR="Avatar" EVENT="OnPlayerChangeUnit(lIndex)" LANGUAGE="JScript">
        var sGloss = Avatar.GetSiGMLUnitName( lIndex);
        sGlossBeingPlayed = sGloss;
</SCRIPT>

<SCRIPT FOR="Avatar" EVENT="OnPlayerChangeMode(bManual)" LANGUAGE="javascript">
        SetButtonsStatus();
</SCRIPT>

<SCRIPT FOR="Avatar" EVENT="OnAvatarInitialised()" LANGUAGE="javascript">
        //LoadPlayList();
</SCRIPT>

<SCRIPT FOR="Avatar" EVENT="OnSigningDataIsAvailable()" LANGUAGE="javascript">
        Play();
        SetButtonsStatus();
</SCRIPT>
<XML id="a">
        <sigml>
                *** sigml should be here ***
        </sigml>
</XML>
```

```
		<XML id="b">
			<sigml>
				*** sigml should be here ***
			</sigml>
		</XML>

		<XML id="c">
			<sigml>
				*** sigml should be here ***
			</sigml>
		</XML>

		<XML id="d">
			<sigml>
				*** sigml should be here ***
			</sigml>
		</XML>
		<XML id="e">
			<sigml>
				*** sigml should be here ***
			</sigml>
		</XML>
		<XML id="f">
			<sigml>
				*** sigml should be here ***
			</sigml>
		</XML>

	<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#99CCCC" ONLOAD="Initialise()"</body>
 <table height=100%  width="100%"     border="1" align="CENTER">
  <tr>
    <td width="90%" height="100%" rowspan="1" align="center" valign="TOP">
	<object classid="clsid:20E52958-E0E3-4C06-8D0E-A477DEFD4933" width = "100%;" height = "100%;" align="right" id="Avatar"></object>
    </td>
  </tr>
  <tr>
  </tr>
  <tr>
    <td align="center" valign="top">
				<input  type="button"  name="btnPlay" value="Play" onClick="Play()">
				<input  type="button"  name="btnStop" value="Stop" onClick="Stop()">
				<input  type="button"  name="btnBack" value="Back" onClick="Back()">
				<input  type="button"  name="btnForward" value="Forward" onClick="Forward()">
    </td>
  </tr>
</table>
</body>
</html>
```

Avatar in a pop-up window In this version, the avatar is only loaded when a signing link is clicked. Again, any number of frames can be used to make up a single web page, but for this version, only one is necessary. You still need a directory index page (usually index.html) giving details of all the html pages used to build the frameset and their layout, together with the javascript functions which will load the avatar.html page into a modeless dialog window when signing is required.

Index.htm (frameset, or top) There is no reason why you cannot change the frameset layout (add frames, etc)

```
<HEAD>
function Play(signSet){
      if(!avatarWindow || avatarWindow.closed){
            avatarWindow=window.showModelessDialog("Avatar.htm",window, "dialogHeight:620px; dialogWidth:350px;
dialogTop:30px; dialogLeft:900px; edge:sunken; scroll:no; status:yes; help:no");

avatarWindow.signSet=signSet;
            //avatarWindow=window.open("Avatar.htm","SigningWindow","width=350,height=700,status=1");
      }
      else{
      avatarWindow.focus();
      avatarWindow.signSet=signSet;
      avatarWindow.Play();
      }
}
</HEAD>
<frameset rows="*">
      <frame name="main" src="Text.htm">
```

The javascript function `Play(signSet)` launches the modeless dialog (called `avatarWindow`) if it is not already loaded. And then calls `avatarWindow.Play()` to sign. Note that this time a variable (`var signSet;`) is set in Avatar.html to indicate which sign sequence ('a' here) is required.

Text.html (mainFrame)

All the first regular page plus image or text links to signing, in the format:

```
<a ref="javascript:window.top.Play('a')">
      <img src="images/Esign_logo03_micro.jpg"
      width="31" height="57" border="0">
</a>
```

(this could all be on one line)

Notice that in `window.top.Play('a')`...
`window.top` refers to the `index.htm` frameset which in turn tells you that the javascript function `Play()` is in `Index.htm`.
`'a'`is a way of referencing the XML that should be played by the avatar for this particular link. Any letter(s) or numbers could be used instead of `'a'`.

Any linked pages will replace this frame, while the modeless dialog remains.

Avatar.htm (modeless dialog)

`<HEAD> .. </HEAD>` containing:
All javascript functions to control the avatar, buttons etc, including `Play()`
(NB Again, in the Initialise() function, order is important:
`BackgroundColour; Initialise(); UserInputBaseURI = C:\MyDir;)`
Also new variable `var signSet;` to store the signing id for the XML

Also include the XML
(or references to it – syntax described elsewhere in this document), eg:
`<XML id="a"> <sigml> <hamgestural_sign gloss="video">` … etc.
note the `"a"` here corresponds to the `'a'` in `Play('a')` in `Text.html`
`<BODY bgcolor="#CCCCFF" ONLOAD="Initialise()">`
…is the start of the <BODY> setting the background colour and calling th `Avatar.Initialise()` function.
The `<BODY> </BODY>` tags also contain the Avatar ActiveX object definition:

```
<object classid="clsid: 20E52958-E0E3-4C06-8D0E-A477DEFD4933"
  width = "100%;"height = "100%;" align="right" id="Avatar"  >
</object>
```

Also include any buttons, slider bars etc which control Avatar play.

2 ADDITIONAL FEATURES

This is not a comprehensive list, but it does address some of the more common “How do I…” questions. Developers are advised to read the ReadMe.txt file in the SiGMLSigning install package, for further information.

2.1 – How to check SiGMLSigning software is installed on a client machine

The following script can be applied:

```
<html>
        <head>
          <title>SiGMLSigning Install Check</title>
          <script language="JavaScript">
            var SSHomeKey = "HKLM\\SOFTWARE\\eSIGN\\SiGMLSigning\\Home";
            var SSStatusMsg = "????";
            function Initialise()
            {
                var shell = new ActiveXObject("WScript.Shell");
                var sshome = "";
                try {
                    sshome = shell.RegRead(SSHomeKey);
                }
                catch (ex) {
                }
                finally{
                    var ssexists = (sshome.length > 0);
                    SSStatusMsg =
                        "SiGMLSigning "+
                        (ssexists ? "exists!" : "does not exist!");
                }
            }
            function ShowSSStatus()
            {
                txtMessage.value = SSStatusMsg;
            }
          </script>
        </head>
        <body onload="Initialise()">
                <h1>SiGMLSigning Install Check</h1>
                <br>
                <textarea id="txtMessage" rows="1"
                    style="width:100%; font:'18pt sans-serif'">
                [[click the button below]]
                </textarea>
                <br>

                <input type="button" name="btnSSShow"
                    value="SiGMLSigning Exists?"
                    onclick="ShowSSStatus()">
                </input>
                <br><hr>
        </body> <html>
```

2.2- How to change the avatar background colour

Place the following line in the html:
`Avatar.BackgroundColour = 0xD0D0A0; -` and you MUST put it BEFORE the line which says:
`Avatar.Initialise(6,H2S)` If you put the new line *after* `Avatar.Initialise(),` the colour won't change.
Change the Hex number to change the actual colour. The `0x` always stays as it is, the 6 characters after that (`D0D0A0`) represent the colour. They go in pairs but the order is BGR, not RGB as you might expect. So:
Blue component = D0 = 208
Green component = D0 = 208
Red component = A0 = 160

2.3 – How to switch ambient motions on and off

Anywhere after Avatar.Initialise(), assuming your avatar is called Avatar, then, if you want to set all the ambients on, in JavaScript :

```
var ambients = (AM_BLINKING  |  AM_BODY  |  AM_HEAD);
Avatar.SetAmbientMotions( ambients );
```

To turn them all off:

```
 var ambients = 0;
Avatar.SetAmbientMotions( ambients );
```

To set just two of them on:

```
 var ambients = (AM_BLINKING  |  AM_BODY);
Avatar.SetAmbientMotions( ambients );
```

Looking at it another way:

```
 var ambients = Avatar.GetAmbientMotions();   gets the three flags in a variable called ambients
ambients |= AM_BODY;                           if you want to switch the body motion *on* and
ambients &= (~ AM_BODY);                       if you want to switch it *off* and then...
Avatar.SetAmbientMotions(ambients);            write it back
```

Replace `AM_BODY` with `AM_HEAD` or `AM_BLINKING` for the other two.

2.4 – How to use alternative directories for SiGML files

Place the SiGML files in your chosen directory.
Set a variable at the top of your (avatar) html file:

```
var USER_BASE_PATH = "C:/Program Files/eSIGN/SiGMLSigning/Files/non-std-UserBase";
```

which is the directory where you keep all your SiGML files for a particular project.
This avoids the files getting mixed in with all the other files in the default C:\Program Files\eSign\SiGMLSigning\Files directory.
Then AFTER (and it has to be after) `ssAvatar.Initialise(avatarID, jSiGMLIn);`
You put the line:

```
ssAvatar.UserInputBaseURI = USER_BASE_PATH;
```

which specifies that directory as an alternative place to look for SiGML files.

2.5 – How to avoid putting all the SiGML in the HTML pages

SiGML files can be referenced by their file names:

```
<sigml> <signing_ref uri='mySigmlFile' format='SiGML'/> </sigml>
```

where `mySigmlFile.sigml` is the name of the SiGML file required.

3 - Reference

3.1 – Available Avatars

In the complete SiGML Signing installation, 8 avatars are available. These are referenced numerically in the web page avatar initialisation script:

```
ssAvatar.Initialise(avatarID, jSiGMLIn);
```

where `ssAvatar` is the name assigned to the avatar instantiation, and avatarID is the numeric variable specifying which avatar is to be used. Possible avatars are:

Avatar	Number
Visia3	1
Visia2	2
Visia4	3
ARPMan	4
NewVisia3	5
VGuido	6
Shaded Vguido	7
Michael	8

The official eSIGN avatar is Vguido, and he should be used in all formal web page applications.

3.2 - SiGMLSigning - Enumerations, Methods, Properties And Events

The SiGMLSigning library provides an interface called `ISiGMLToSigningAvatar` which has the following enumerations, methods, properties and events. These can be scripted into web pages to access various aspects of avatar functionality. A few recent additions are not included here.

Enumerations:

`enum {NoAvatar, Visia3, Visia2, Visia4, ARPMan, NewVisia3, Guido} Avatar;`

Used with the `Initialise` and `SwitchAvatar` methods to denote which avatar to use.

`enum {Manual, Auto} PlayerMode;`

Used with the `PlayerChangeMode` method.

Methods:

```
Initialise( [defaultvalue(Guido)] Avatar a,
                     [in,defaultvalue(NULL)] IDispatch* pJSiGMLInput);
```

Once an instance of a `ISiGMLToSigningAvatar` has been instantiated call this method to specify the avatar to use and to pass in a pointer to a `JSiGMLIn` library.

`Terminate();`

This does not do anything now: ignore.

`ResetCamera();`

This method restores the camera to its default startup position.

`AcceptNewSiGMLURI( [in] BSTR bsURI, [in] int nFrameRate);`

Call this to pass in an URI to a SiGML file for which animation frames are required.

`AcceptNewSiGMLString( [in] BSTR bsSiGMLString, [in] int nFrameRate);`

Call this to pass in a SiGML string for which animation frames are required.

`GetSiGMLUnitSiGML([in] long lIndex, [out, retval] BSTR* bsSiGML);`

Call this to obtain the hamgestural_sign text of the Signing Unit identified by `lIndex`.

`GetSiGMLUnitHNSSign([in] long lIndex, [out, retval] BSTR* bsHNS);`

Call this to obtain the `hns_sign` text (if any) of the Signing Unit identified by `lIndex`.

`GetSiGMLUnitName([in] long lIndex, [out, retval] BSTR* bsName);`

Call this to obtain the name (either gloss of file name) of the Signing Unit identified by `lIndex`.

`GetSiGMLUnitFrameRange([in] long lIndex, [out] long *plFirstFrame,[out] long* plLastFrame);`

Call this to obtain the first and last indices of the animation frames of the current frame range of the Signing Unit identified by `lIndex`.

`GetSiGMLUnitIndex([in] long lFrameIndex, [out, retval] long* plIndex);`

Call this to obtain the Signing Unit index of the given animation frame index, `lFrameIndex`.

`PlayerChangeMode([in]PlayerMode pm);`

Change the player mode.

```
PlayerSetFrameRange( [in] long lFirstFrame, [in] long lLastFrame);
```
Sets the current animation frame range.
```
PlayerShowFrame([in] long lFrameIndex, [out, retval] BOOL *pVal);
```
Causes an animation frame, identified by `lFrameIndex`, to be rendered by an avatar.
```
PlayerShowNextFrame();
```
Causes the next animation frame in the list animation frames to be rendered by an avatar.
```
PlayerShowPreviousFrame();
```
Causes the previous animation frame in the list animation frames to be rendered by an avatar.
```
PlayerShowNextSignUnit([in] BOOL bSetSignFrameRange);
```
Causes the first animation frame of the next Signing Unit in the list of Signing Units to be rendered by an avatar.
```
PlayerShowPreviousSignUnit([in] BOOL bSetSignFrameRange);
```
Causes the first animation frame of the previous Signing Unit in the list of Signing Units to be rendered by an avatar.
```
SetDefaultFrameRange();
```
Sets the current animation frame range to the default values: the first animation frame in the list and the last animation frame in the list.
```
GetCASFrame([in] long lFrameIndex,
  [out] float* pfpTimeStamp,
       [out] SAFEARRAY(VARIANT)* psaBoneLengths,
            [out] SAFEARRAY(VARIANT)* psaBoneNames,
                  out] SAFEARRAY(VARIANT)* psaBoneValues,
                        [out] SAFEARRAY(VARIANT)* psaMorphNames,
                              out] SAFEARRAY(VARIANT)* psaMorphValues);
```
Retrieve all the CAS data for the animation frame identified by `lFrameIndex`.
```
GetSiGMLUnitExplicitSiGML([in] long lFrameIndex,  [out, retval] BSTR* bsExplicitSiGML);
```
Call this to obtain the Explicit SiGML of the Signing Unit identified by `lIndex`.
```
SwitchAvatar(Avatar aNew);
```
Call this to load a different avatar.

Properties:
```
AppInputBaseURI - string - read/write
```
This specifies a folder in which `signing_ref` files can be found.
```
UserInputBaseURI - string - read/write
```
This specifies a another folder in which `signing_ref` files can be found.
```
DoBlendFrames - boolean - read/write
```
If True then blend region animation frames are created if required. If False no blend region animation frames are created.
```
SigningDataAvailable - boolean - read only
```
True when all Signing Units and animation frames have been generated for a SiGML input file (or string)
```
CountSiGMLUnits - long - read only
```
The count of Signing Units.

`CountFrames - long - read only`

The count of animation frames generated.

`PlayerDoCycleFrames - boolean - read/write`

If True, after playing the last animation frame the next animation frame to be played is the first animation frame. If in `Manual` mode, and calling `PlayerShowPreviousFrame()` after playing the first animation frame the next frame to be played is the last animation frame.

`PlayerSpeedLog2 - float - read/write`

For adjusting the speed at which the avatar plays animation frames.

`PlayerModeIsManual - boolean - read only`

True if `Manual` mode, False otherwise.

`PlayerFrameLo - long - read only`

The first animation frame in the current frame range.

`PlayerFrameHi - long - read only`

The last animation frame in the current frame range.

`PlayerFrameIndex - long - read only`

The index of the last played animation frame.

`PlayerUnitIndex - long - read only`

The index of the last played Signing Unit

`PlayerShowPreviousFrameOK - boolean - read only`

If True, it is possible to show the previous frame in the current frame range.

`PlayerShowNextFrameOK - boolean - read only`

If True, it is possible to show the next frame in the current frame range.

`PlayerShowAdjacentUnitOK - boolean - read only`

If True, it is possible to show the first animation frame of the next or previous Signing Unit.

`PlayerSetFrameRangeOK - boolean - read only`

If True, it is possible to set a current frame range.

`Version - string - read only`

Animgen version string.

`AnimgenLogFile - string -read only`

URI of log file used by animgen.

`AppendToLogFile - boolean - write only`

If True, the animgen output for the current signing session is appended to the output from previous signing sessions. If False, the previous contents of the animgen log file are discarded.

`CanSwitchAvatar - boolean - read only`

If True, a call to `SwitchAvatar()` should succeed; if False a call to `SwitchAvatar()` will not succeed.

Events:

`OnAvatarInitialised()`

An avatar has been successfully loaded. This event is fired after a call to `Initialise()` or `SwitchAvatar()`.

`OnStartSiGMLIn()`

The processing of the SiGML file (or string) has started.

`OnSiGMLInUnit(long lIndex)`

The Signing Unit specified by the `lIndex` parameter has been received.

`OnFinishSiGMLIn(BOOL bSuccess, BSTR bsStatus)`

The processing of the SiGML file (or string) has finished.

`OnStartSigningRef(BSTR sURI, long lIndex)`

A `signing_ref` has been found and is now being processed.

`OnFinishSigningRef()`

A `signing_ref` has finished being processed.

`OnStartGenFramesForUnit(long lIndex, long lFrameCount, BSTR bsStatus)`

The generation of `lFrameCount` animation frames has started for the Signing Unit denoted by `lIndex`.

`OnGenFrameForUnit(long lIndex, long lFrameIndex)`

An animation frame, denoted by `lFrameIndex`, has been generated for the Signing Unit denoted by `lIndex`.

`OnFinishGenFramesForUnit(long lIndex, BOOL bSuccess, BSTR bsStatus)`

The generation of animation frames for the Signing Unit denoted by `lIndex` has completed.

`OnPlayerChangeMode(BOOL bManual)`

The playing mode has changed.

`OnPlayerStartShowFrame( long lFrameIndex, long lStartTimeMillis)`

The rendering of an animation frame by an avatar, denoted by `lFrameIndex`, has started. `lStartTimeMillis` denotes the start time.

`OnPlayerFinishShowFrame( long lFrameIndex, long lFinishTimeMillis)`

The rendering of an animation frame by an avatar, denoted by `lFrameIndex`, has finished. `lFinishTimeMillis denotes the finish time.`

`OnPlayerChangeUnit(long lIndex)`

This occurs when an animation frame is played that belongs to a different Signing Unit to the previous animation frame played.

`OnSigningDataIsAvailable()`

All Signing Units have been determined and all animation frames have been generated for the SiGML file (or string) passed in as input.

`OnAnimgenLogFileAvailable()`

The log file used by animgen is available to be viewed.

`OnAnimgenLogFileNotAvailable()`

The log file used by animgen is not available to be viewed - animgen is still using it.