



# Deliverable D1.5

## Browser extension for speech feedback

Contract number: **FP7-216529** PinView

Personal Information Navigator Adapting Through Viewing

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under *grant agreement* n° 216529.



## Identification sheet

<b>Project ref. no.</b>	<b>FP7-216529</b>
<b>Project acronym</b>	PinView
<b>Status and version</b>	Final, Revision: 2.5
<b>Contractual date of delivery</b>	31.12.2008
<b>Actual date of delivery</b>	27.06.2009
<b>Deliverable number</b>	D1.5
<b>Deliverable title</b>	Browser extension for speech feedback
<b>Nature</b>	Prototype
<b>Dissemination level</b>	PU – Public
<b>WP contributing to the deliverable</b>	WP1 Forms and protocols for enriched relevance feedback
<b>Task contributing to the deliverable</b>	Task 1.5 Implementation of point-and-speak feedback
<b>WP responsible</b>	Teknillinen korkeakoulu
<b>Task responsible</b>	Teknillinen korkeakoulu
<b>Editor</b>	Jorma Laaksonen <jorma.laaksonen@tkk.fi>
<b>Editor address</b>	P.O.BOX 5400, FI-02015 TKK, Finland
<b>Authors in alphabetical order</b>	Mikko Kurimo, Jorma Laaksonen, Teemu Ruokolainen
<b>EC Project Officer</b>	Pierre-Paul Sondag
<b>Keywords</b>	browser extension, enriched relevance feedback, audio data, speech recognition, AJAX, XMLHttpRequest
<b>Abstract</b>	This report describes the implementation of a web browser extension that collects audio (speech) signal captured with a microphone while displaying a document in the browser. The extension is available for Linux and Windows platforms and has been implemented by using a C++ program connected to the Mozilla Firefox browser through Mozilla's XP-COM interface. In addition there is a JavaScript program that handles the transmission of the recorded data to a collecting server. This latter part shares some of the code with the browser extension that was implemented in PinView Task 1.4 for the collection of pointer movement and keyboard event data. The communication between the client and server is based on the AJAX technology, where XML formatted content is asynchronously exchanged by using the XMLHttpRequest method.

## List of annexes

pinview-1.00.xpi – XPI package containing the software of PinView extension, v. 1.00, available at <http://www.pinview.eu/extension/>

## Contents

<b>1 Overview</b>	<b>4</b>
<b>2 Introduction</b>	<b>5</b>
<b>3 Implementation</b>	<b>5</b>
3.1 JavaScript class structure . . . . .	5
3.2 PinViewAudio C++ class and its XPCOM interface . . . . .	7
3.3 portaudio library and its use . . . . .	8
3.4 Format and size of audio data packets . . . . .	8
3.5 File structure . . . . .	9
3.6 Software prerequisites . . . . .	10
<b>4 Discussion</b>	<b>10</b>

# 1 Overview

This Deliverable D1.5 of the *Personal Information Navigator Adapting Through Viewing*, PinView, project, funded by the European Community's Seventh Framework Programme under Grant Agreement n° 216529, constitutes the output of Task 1.5 *Implementation of point-and-speak feedback*. The content of the original version (Revision 1.30) of this deliverable was found insufficient in the first periodic review of the PinView project. As requested by the reviewers, technical details of the implementation have now been added.

The deliverable aims at providing a sufficient detail on implementing the audio or point-and-speak feedback, which belongs to one of the four user interaction modalities used as *enriched relevance feedback* that have been earlier planned in the project. The other modalities that are planned for extracting enriched relevance feedback, as stated in PinView's previous Task 1.3 *Definition of transport protocol for enriched feedback* [2], are *eye movements*, *pointer movements and events* and *keyboard events*.

The audio feedback has been implemented by programming a browser extension in the Mozilla Firefox browser by using C++ programming, Mozilla's XPCOM programming interface and JavaScript language. For the recording of microphone signals, we have used the *portaudio* C library that is freely available and can be used on both Linux and Windows platforms.

Deliverable D1.5 was in the PinView Description of Work tentatively named as *Browser applet or plug-in for point-and-speak feedback*, but due to the actual nature of the implemented software, it has now been renamed as *Browser extension for speech feedback*.

The implementation work described in this report was coordinated with the concurrent Task 1.4 *Implementation of pointer track feedback* and as a result, the two browser extensions have been merged into one.

## 2 Introduction

The goal of this report is to describe the implementation of audio or point-and-speak feedback that can be transmitted from a web browser to a content-based image retrieval (CBIR) server. Based on the general transport framework of PinView Task 1.3 *Definition of transport protocol for enriched feedback* [2], a browser extension has been programmed to collect client's audio signal captured with a microphone. The extension has been combined with that of PinView Task 1.4 *Implementation of pointer track feedback* [3] into one extension that can handle three input modalities: audio, pointer movements and keyboard events. Later also the recording of eye movements will be implemented in the same framework.

Extensions can in general be used to add new features that extend the functionality of a web browser such as Mozilla Firefox<sup>1</sup>, which is a free and open source web browser with great popularity today. The implementation work detailed in this report follows the instructions on building extensions in Mozilla Developer Center (MDC)<sup>2</sup>, the official Mozilla Foundation website for developing Mozilla applications. Especially important has been the combination of C++-programmed audio processing methods with the Mozilla browser through Mozilla's XPCOM programming interface<sup>3</sup>.

The implemented browser extension enables recording audio signal data and storing it in proper XML-formatted packets. The extension then transfers the data to the search server asynchronously over XMLHttpRequest protocol<sup>4</sup> defined by the World Wide Web Consortium (W3C)<sup>5</sup> and often used in implementing *Asynchronous JavaScript and XML* (AJAX) type of asynchronous content updates in the web.

The rest of this report is organised as follows. In Section 3, we explain the class structures of the JavaScript and C++ implementations, describe the use of the `portaudio` library, analyse the size of the audio data packets, list the files that constitute the audio-related part of the PinView extension and state the software prerequisites of installing and using the audio-enabled extension.

In Section 4, we present discussion on the modifications applied to the system design relating to the use of audio data. In addition, we discuss some aspects of the speech recognition task executed on the server side.

## 3 Implementation

In this section we will specify the principles of implementing the audio part of the PinView browser extension. As stated in PinView's Task 1.3 *Definition of transport protocol for enriched feedback* [2], the user's web browser will in general need to be equipped with a special add-on or extension that is capable of interpreting the `<meta>` tags in the server's messages. Then the browser extension needs to record data of the specified user interaction modalities and store them in proper XML formatted packets. Finally, it has to communicate the recorded data over the XMLHttpRequest protocol to the search server. The XMLHttpRequest protocol is already available in all modern web browsers including Internet Explorer, Firefox, Opera and Safari.

### 3.1 JavaScript class structure

The modalities of user interaction concerned in PinView Deliverable D1.4 [3] include *pointer movements and events* and *keyboard events*. The JavaScript class inheritance of those input

---

<sup>1</sup><http://www.mozilla.com/en-US/firefox/>

<sup>2</sup><https://developer.mozilla.org/En>

<sup>3</sup><https://developer.mozilla.org/en/XPCOM>

<sup>4</sup><http://www.w3.org/TR/XMLHttpRequest/>

<sup>5</sup><http://www.w3.org/>

modalities has been complemented by adding the *audio* modality and deriving it from the *Modality* base class in a manner similar to that of the other modalities and depicted in Figure 1. The roles of the JavaScript classes are detailed as follows:

**Modality** is defined to be the base class of the classes for pointer movements, keyboard events, audio recordings and eye tracking data. These all collect their input and store it internally in the `xmlDoc` object which is then periodically transmitted to the server URL.

**Audio** is defined to be a sub-class of *Modality*. This means that it inherits all the functions defined in *Modality*, but also adds its own methods for audio-specific functionality.

The method `handleTimeout()` is the most important new method in the *Audio* class as it is called by a JavaScript timer. In the current implementation the timer call is made every second. In its first invocation, the `handleTimeout()` method calls `initializeXPCOM()`. If that call has been successful, `readAndSendOneAudioBuffer()` is called in all invocations. These and other *Audio* class methods of interest are as follows:

`initializeXPCOM()` creates and initialises an instance of C++ *PinViewAudio* class by using Mozilla's XPCOM programming interface. The initialisation of the C++ module is performed by calling the `PinViewAudio::Initialize()` method after constructing the *PinViewAudio* object.

`readAndSendOneAudioBuffer()` requests from the *PinViewAudio* class object all audio data packets recorded after the previous call. The data is transferred in the Base64-encoded format, stored in an XML document and sent to the server by using the `SendXML()` method.

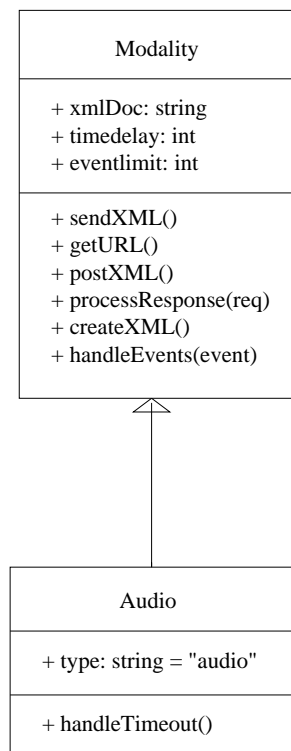


Figure 1: Class diagram of the relationship between base class *Modality* and its sub-class *Audio*.

`initializeAudio()`, `readAudio()` and `audioStatus()` are utility functions for debugging and testing purposes. They can be bound to some keyboard events for controlling and testing the audio functionality of the PinView extension.

### 3.2 PinViewAudio C++ class and its XPCOM interface

The collection of the audio data from a microphone has been implemented by a module written in C++. The module consists of class `PinViewAudio` and its XPCOM interface<sup>6</sup> named `IPinViewAudio`. These classes can be compiled with the GNU g++ compiler<sup>7</sup> version 4.3.2 for the Linux operating system and with the free Microsoft Visual Studio 2008 Express Edition compiler<sup>8</sup> for the Windows operating systems.

The implementation is compiled and linked together with the Mozilla Software Development Kit (SDK), either version `gecko 1.8.0.4`<sup>9</sup> or `xulrunner 1.9.0.5`<sup>10</sup>. The former can be used together with Firefox versions 2.x and 3.x whereas the latter only with versions 3.x. For the sake of wider applicability, the publicly available extension has been linked with the `gecko 1.8.0.4` SDK. In the development and testing of the extension, various versions of Firefox 3.0.x browser have been used in both Linux and Windows systems.

The XPCOM interface class `IPinViewAudio` is defined as follows:

```
[scriptable, uuid(9ad272c1-283e-4bc6-944a-d4a47e913009)]
interface IPinViewAudio : nsISupports
{
    boolean Initialize(out ACString msg);

    ACString Status();

    long DataAvailable();

    ACString ReadBase64(out ACString time, out float duration);

    ACString CatchUp();
};
```

The methods are:

`Initialize()` initialises the internal data structures and the `portaudio` library. Recording of microphone audio is started.

`Status()` returns a string that tells the versions of the software, the compiler and the `portaudio` library, the number of audio buffers allocated, the size of each buffer, the counts of buffer writes and reads. In error situations also the counts of oversize buffers, buffer overruns and drops are reported.

`DataAvailable()` returns the size of available audio data in samples.

`ReadBase64()` collects all available audio data from internal buffers and returns them in a Base64-encoded ASCII string. The internal buffers are released for further use.

`CatchUp()` neglects all stored audio data and releases the internal buffers.

<sup>6</sup><https://developer.mozilla.org/en/XPCOM>

<sup>7</sup><http://gcc.gnu.org/>

<sup>8</sup><http://www.microsoft.com/Express/vc/>

<sup>9</sup>[https://developer.mozilla.org/En/Gecko\\_SDK](https://developer.mozilla.org/En/Gecko_SDK)

<sup>10</sup><https://developer.mozilla.org/En/XULRunner>

The internal working of the `PinViewAudio` class is such that it keeps a ring of pointers to preallocated memory blocks for storing the audio data. In the current implementation there are 100 such memory blocks, each capable of holding 10000 bytes or 5000 audio samples. Each time when new audio data are available from the `portaudio` library, a memory block from the “tail” of the ring is used to store them. The “tail” pointer is then incremented by one. When data are requested from the Mozilla browser (currently once every second) as a consequence of the JavaScript timer timeout, all filled memory blocks in the “head” of the ring are returned in Base64-encoded form. The “head” pointer is then skipped to the first memory block not yet returned.

### 3.3 portaudio library and its use

The microphone audio data is collected by using the `portaudio` library<sup>11</sup>. It is available for free and can be used on both Linux and Windows platforms. The version used in the current PinView extension is `v19_20071207`. On the Linux platform we assume that the `portaudio` library has been installed in the system, whereas for the Windows platform it is included in the PinView extension.

The audio data is recorded with the following parameters of the `portaudio` library:

**Number of channels** is one, i.e. the recordings are monophonic.

**Sample format** is `paInt16`, i.e. 16-bit integer values, each taking two bytes.

**Sample frequency** is 16 kHz.

**Streaming mode** is `callback`, i.e. the library issues a callback to the `PinViewAudio` class when it is ready to deliver a new packet of input data.

### 3.4 Format and size of audio data packets

The *enriched relevance feedback* (erf) transmission framework and format specified in PinView Task 1.3 [2] has been in this work extended and applied to audio data. Figure 2 gives an example of an erf message that contains audio data of duration of approximately one second. The actual audio data is Base64 encoded and has been dropped from the example.

The Base64 encoding is necessary for including the otherwise binary data in the XML format message. Due to the encoding, the byte size of the data is expanded by the factor of  $8/6 = 1.33$ . As the size of the other parts of the XML message are negligible compared to that of the encoded audio data, the total audio data flux from the browser extension to the server can be estimated to be

$$1.33 \cdot 16 \text{ kHz} \cdot 2 \text{ byte} \approx 43\text{KB/s} .$$

This size could be approximately halved if standard gzip compression<sup>12</sup> were applied in the stream. In one of our experiments a Base64-encoded XML audio file of size 42263 bytes was reduced to 21700 bytes by using gzip. Another source of data size reduction could come from using some efficient lossless or lossy audio compression scheme before the Base64 encoding stage. Both of these approaches are however bound to increase the computational cost on the client side and also lengthen the lag in audio data transfer.

---

<sup>11</sup><http://www.portaudio.com/>

<sup>12</sup><http://www.gzip.org/>

```

<?xml version="1.0"?>
<stream.start>
  <erf href="http://192.168.2.6:5611/query/Q:090627:003521:6791:0"
    type="audio" scheme="pinview/1.0"/>
  <data>
    <audio>
      <timestamp>Fri, 26 Jun 2009 21:35:29.270738 GMT</timestamp>
      <duration>980.375</duration>
      <messagetype>AUDIO</messagetype>
      <urgent>0</urgent>
      <waveform rate="16000" format="16bit" channels="1"
        encoding="base64" status="pinviewaudio_impl_cpp=$Revision: 1.24
        $ MOZILLA_VERSION=1.8.0.4 GCC=4.3 PA=&quot;PortAudio V19-devel
        (built Mar 4 2009)&quot;; buf_count=100 buf_size=10000
        in_buf_idx=4877 out_buf_idx=4877 maxsize=342 oversize=0
        overrun=0 drop=0"> Base64-encoded data removed </waveform>
    </audio>
  </data>
</stream.start>

```

Figure 2: Content of an XMLHttpRequest with audio data.

### 3.5 File structure

As the audio extension is implemented by using both JavaScript and C++ its file structure is more evolved than that of the pointer data extension developed in PinView Task 1.4. In addition to the files mentioned in [3], the files listed in Figure 3 have now been included in the PinView extension file `pinview-1.00.xpi`. The JavaScript archive file `chrome/pinview.jar` further contains the JavaScript program file `chrome/content/audio.js`. The `pinviewaudio.xpt` files contain the compiled XPCOM interface definitions and the `pinviewaudio.dll` and `pinviewaudio.so` files the implementation of the PinViewAudio C++ class for the Windows and Linux operating systems, respectively. In Windows, two additional libraries are needed. First the `portaudio_x86.dll` library that contains the `portaudio` library and then the `bdsStubLoader.dll` library that is needed for ensuring the correct run-time loading of the `portaudio` Windows library.

```

pinview-1.00.xpi:
  chrome/pinview.jar
  platform/WINNT/components/pinviewaudio.xpt
  platform/WINNT/components/bdsStubLoader.dll
  platform/WINNT/libraries/pinviewaudio.dll
  platform/WINNT/libraries/portaudio_x86.dll
  platform/Linux/components/pinviewaudio.xpt
  platform/Linux/components/pinviewaudio.so
  README.portaudio

```

Figure 3: Audio-related files in the `pinview-1.00.xpi` file.

### 3.6 Software prerequisites

The extension has been designed for and tested in two operating systems, Windows XP with Service Pack 3<sup>13</sup> and Ubuntu Linux release 8.10<sup>14</sup>. In both systems, various versions of Mozilla Firefox 3.0.x<sup>15</sup> have been used together with the extension.

In Ubuntu Linux, it is assumed that the `libportaudio2` software package has been installed. As the `portaudio` library is readily included in the extension for the Windows operating systems, it does not need to be pre-installed. (In fact, there is no standard `portaudio` installation available for Windows, if there were we had used it.) In order to be able to use the extension, Windows users on the other hand need to first install Microsoft Visual C++ 2008 Redistributable Package<sup>16</sup> which is also available on the PinView extension download page<sup>17</sup>.

## 4 Discussion

Here we address two aspects of the project development relating to the use of audio information. Firstly, the change in the nature of the point-and-speak task and secondly, the expected extent of performance of the speech recognition system at the server-side.

The original naming of the speech feedback task (*Implementation of point-and-speak feedback*) implies that the data stream of the `audio` should be dependent on the data stream of the `pointer`. The dependency here would suggest that the audio data should always be somehow connected to the pointer data, e.g. every audio sample would also include the information of the pointer location at the same time instance. However, the data streams of these two modalities have now been completely separated from each other, i.e. now the audio data is considered as autonomous as the other modalities. Hence the connecting attributes between all the data streams are the time stamps which tell us the time instances the samplings have occurred. By this modification we obtain more freedom as to connecting the information retrieved from the modalities. For example, the point-and-speak concept as described above is still fully implementable. However, the system is now more robust in the sense, that e.g. the pointer data can, in principle, be discarded without this affecting the handling of the audio and eye movement data.

As for the performance of the speech recognition system at the server side, it should be notified, that the current state-of-the-art recogniser [1] does not provide a real-time, continuous recognition result. The results (multiple word arrays with different probabilities) are produced in segments with varying lengths. However, this does not have to be considered a severe weakness, if we are willing to accept the few second delays and simply take the recognition results into account in the following processing as soon as they are made available.

Furthermore, the multimodal environment gives rise to an interesting addition to the speech recognition processing. From the pointer and eye movement data we can derive information about not only *what* is on the screen, but also *what is it there the user is likely to be interested in*. This additional knowledge can then be utilised in recognition with a technique called *topic adaptation* which is further described via the following brief example. Let us assume we know the user has focused his attention at a screen, specifically on a picture depicting a harbour. Now, when commenting on his view, it is presumable that he will more likely utter the word “ship” than “chip”. This additional information can provide the recogniser crucial benefit when it is trying to make a decision between the two phonetically similar

<sup>13</sup><http://www.microsoft.com/windows/windows-xp/>

<sup>14</sup><http://www.ubuntu.com/>

<sup>15</sup><http://www.mozilla.com/en-US/firefox/>

<sup>16</sup><http://www.microsoft.com/downloads/details.aspx?FamilyID=9b2da534-3e03-4391-8a4d-074b9f2bc1bf>

<sup>17</sup><http://www.pinview.eu/extension/>

words. This topic adaptation is implemented as a modification of the n-gram model weights in the statistical language model used by the recogniser. The topics and adaptation weights are pre-determined by training with the existing descriptions of the training data available for the application and switched on based on any existing or hypothesised descriptions or cues about the view the user is currently viewing.

## Acknowledgements

Collaborators in the PinView project, especially Bernhard Lackner of celum, are acknowledged for their valuable comments and contributions concerning the content of this report. The reviewers of the PinView project's first periodic review, Prof. Christian Bauckhage of Fraunhofer Institute for Intelligent Analysis and Information systems and Prof. Fulvio Corno of Politecnico di Torino, are thanked for their expert comments and guidance in improving this report.

## References

- [1] Teemu Hirsimäki, Mikko Kurimo, and Janne Pylkkönen. Importance of high-order n-gram models in morph-based speech recognition. *Transactions on Audio, Speech and Language Processing*, 2009. Accepted for publication.
- [2] Jorma Laaksonen. Definition of transport protocol for enriched feedback. PinView FP7-216529 Project Deliverable Report D1.3, September 2008. Available online at <http://www.pinview.eu/deliverables.php>.
- [3] He Zhang, Mats Sjöberg, and Jorma Laaksonen. Browser extension for pointer track feedback. PinView FP7-216529 Project Deliverable Report D1.4, December 2008. Available online at <http://www.pinview.eu/deliverables.php>.