



## APIDIS

Autonomous Production of Images based on Distributed and Intelligent Sensing

STREP Project, 1st FP7-216023

### **D6.1 Software environment definition through release of test-bed for flexible access to content**

Due date of deliverable: 31-12-2008

Actual submission date: 08-01-2009

Start date of project: 1<sup>st</sup> January, 2008

Duration: 36 months

Lead contractor for this deliverable: ACIC

[Revision Final v1]

<b>D6.1</b>	<b>Software environment definition through release of test-bed for flexible access to content</b>
Project Acronym :	APIDIS
Contract No :	FP7-216023
Due Date :	31-12-2008
Reply To:	Christophe Parisot <a href="mailto:parisot@acic-tech.be">parisot@acic-tech.be</a>
Actual date of delivery	08-01-2009

## 1 Executive Summary

---

This document discusses the platform that is used for the integration of the different APIDIS software modules. It describes the test-bed for personalized and interactive access to content.

**Deliverable Identification Sheet**

<b>Project ref. no.</b>	FP7-216023
<b>Project acronym</b>	APIDIS
<b>Project full title</b>	FP7-216023
<b>Security (distribution level)</b>	Public (PU)
<b>Contractual date of delivery</b>	Month 12, 12, 31, 2008
<b>Actual date of delivery</b>	Month 12
<b>Deliverable number</b>	D6.1
<b>Deliverable name</b>	Software environment through release of test-bed for flexible access to content
<b>Type</b>	Report
<b>Status &amp; version</b>	Final
<b>Number of pages</b>	15
<b>WP / Task responsible</b>	WP6 / ACIC
<b>Other contributors</b>	UCL, EPFL
<b>Main Author(s)</b>	Christophe Parisot, Christophe De Vleeschouwer, Yannick Boursier
<b>EC Project Officer</b>	Albert Gauthier
<b>Abstract</b>	This document discusses the platform that is used for the integration of the different APIDIS software modules. It describes the test-bed for personalized and interactive access to content.
<b>Keywords</b>	Test-bed.
<b>Sent to peer reviewer</b>	UCL
<b>Peer review completed</b>	yes
<b>Circulated to partners</b>	yes
<b>Read by partners</b>	yes
<b>Mgt. Board approval</b>	Pending

Table of contents

<b>1 EXECUTIVE SUMMARY</b>	<b>2</b>
<b>2 INTRODUCTION</b>	<b>5</b>
<b>3 TEST-BED FOR PERSONALIZED AND INTERACTIVE ACCESS TO CONTENT</b>	<b>6</b>
<b>GENERAL DEVELOPMENT PLATFORM</b>	<b>6</b>
<b>FLEXIBLE RANDOM ACCESS CONTENT STORAGE</b>	<b>6</b>
<b>COMMON APIDIS DEVELOPMENT UTILITIES</b>	<b>7</b>
<b>CURRENT TEST-BED FOR INTERACTIVE ACCESS TO CONTENT</b>	<b>7</b>
<b>ANNOTATION TOOLS</b>	<b>10</b>
<b>GENERATION OF PLANAR IMAGES FROM OMNIVISION</b>	<b>10</b>
<b>4 CONCLUSIONS</b>	<b>13</b>
<b>5 REFERENCES</b>	<b>14</b>

## 2 Introduction

---

This document presents the basics of the platform that has been developed to support the integration of the different APIDIS software modules.

It describes the core kernel of the test-bed, and explains how it has been augmented for personalized and interactive access to content.

The development of the integrated platform is supposed to be incremental, and will benefit from the modules designed by each partner all along the project.

### 3 Test-bed for personalized and interactive access to content

---

Involved partners: ACIC, UCL, EPFL

The test-bed collects the contributions of all partners into a common environment with the goal of testing the concepts developed within the project.

#### ***General development platform***

In order to ease the integration of the different modules, a common development platform has been prepared and distributed to partners. It consists in:

- Ubuntu 8.04 virtual machine that can be played with the free VMware Player application from VMware.
- gcc-3.4 and g++-3.4 c/c++ compilers.
- WxWidgets 2.8 graphical development toolkit.
- FFmpeg encoding/decoding library

#### ***Flexible random access content storage***

This section is taken from deliverable D3.1 [1]. It describes the way APIDIS data are stored.

In order to get flexible and generic access to all Apidis files, the following directory structure and files naming conventions are used.

Any information that is stored is considered as a media. Videos, sounds, low level features, high level metadata and all other information follow the same storage and access rules.

A media can be uniquely determined by a name (e.g. camera1) and a type (e.g. objects.xml).

Since some media cannot be saved within single files (e.g. several hours of videos), the filenames include a time stamp following the ISO 8601 date/time syntax. In general, the time stamp of each file refers to the time stamp of the first data it includes (e.g. time stamp of the first frame in a given video file). Time stamps in filenames allow for fast retrieval of media data at a given period.

Then, filenames look like:

MediaName\_ISOTimeStamp.MediaType

Since some media may be split into a large amount of files, files are organized in a directory tree structure that reduces the amount of files per sub-directory and also optimizes the search of a media at a given period of time. Indeed, the tree structure which is implemented is:

RootMediaDirectory / YYYY / MM / DD / hh / mm / ... file

where “/” splits a directory into sub-directories, YYYY is the year, MM the month, DD the day in the month, hh the hour (in 24 hours format), mm the minutes, etc. When looking for a particular date, it is very fast to locate the file that is associated with this date.

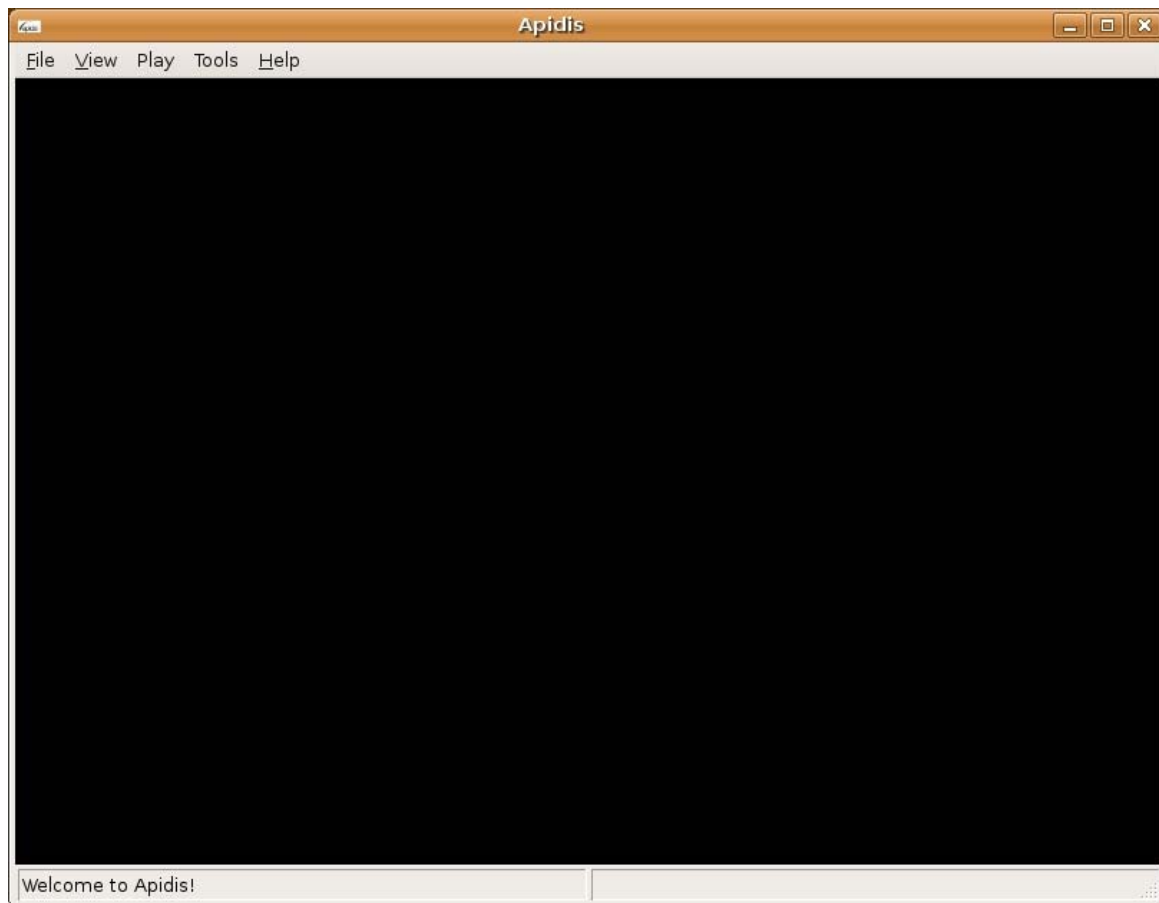
### ***Common APIDIS development utilities***

The following modules have been developed and distributed to partners in order to factorize recurrent tasks:

- ApidisTime class: this c++ class provides the basic functions to manage time stamps and the ISO8601 date/time syntax.
- ApidisImage class: this c++ class is the container for pictures. It currently supports the YUV420 planar image format and will be extended in case other formats are required by partners.
- ApidisVideoDecoder class: this c++ class encapsulates the mechanisms to ask for particular frames (next one, previous one, at a particular instant, etc.) in a recorded video stream with abstraction of the physical storage (location of files, codec, etc.)
- MvMediaFinder class: this c++ class provides high level functions to retrieve a file of a given media at a given time stamp. It also provides the file name that is to be used to write a given media at a given time stamp. This class aims at abstracting the organization of media files on the disk.
- ApidisModule class: this c++ class abstracts a thread that can exchange messages with the other APIDIS modules. The function to run when the thread receives messages and the function to run in the thread loop can be defined. An ApidisModule has also functions to send pictures to the display (see next bullet).
- ApidisDisplayWindow class: this c++ class inherits from a wxWindow object. It can also be considered as a module which displays the graphical elements sent by others modules (e.g. frames).
- ApidisModulePlayer class: this c++ class provides the features for playing back the APIDIS recorded video dataset. It is an example that shows how to easily develop a complete task using most of the previous utilities.
- common wxConfig object: this object allows for the storage of global options (e.g. location of videos, period to consider, etc.). Those data are automatically reloaded when the application is restarted.

### ***Current test-bed for interactive access to content***

The test-bed is a windowed application that currently allows for playing back the recorded video streams. In the near future, it will be enriched with the partners' contributions. Figure 1 to 4 show some snapshots of the test-bed in its current version.



**Figure 1: Apidis application when launched**

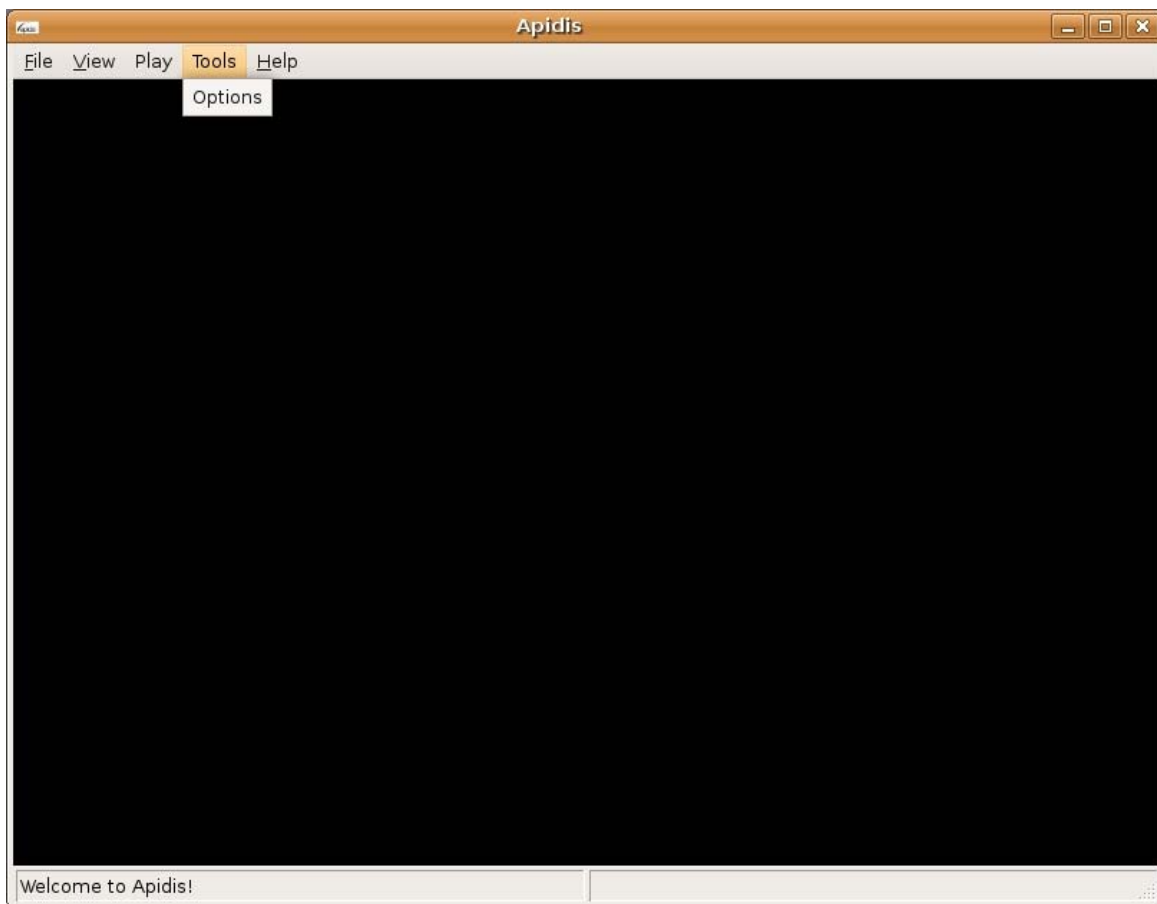


Figure 2: Apidis options menu

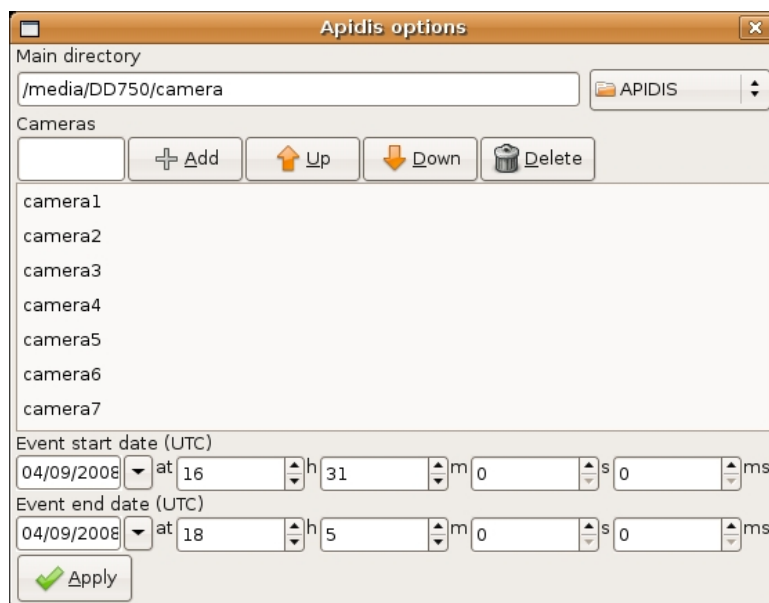


Figure 3: Apidis global options

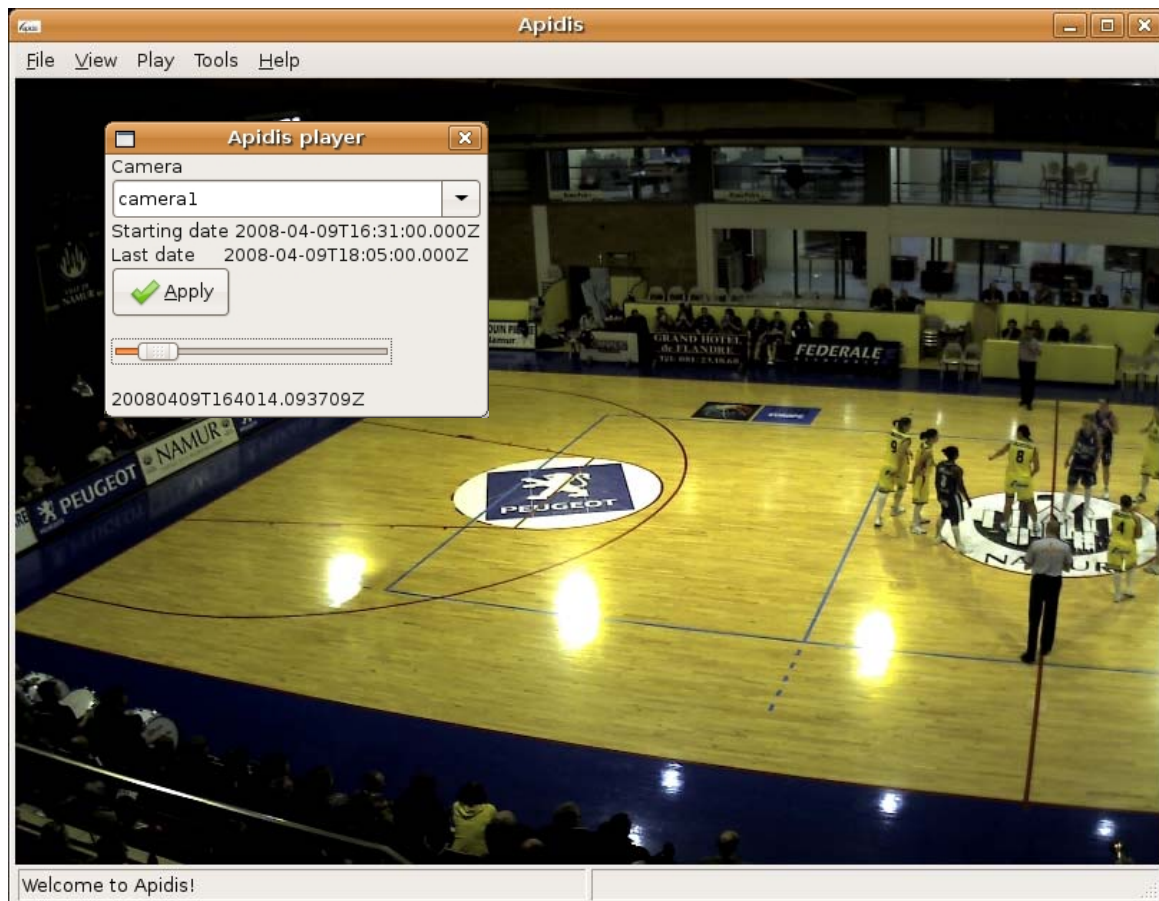


Figure 4: Apidis player window

### ***Annotation tools***

The first module that has been designed to augment the testbed consists in a user-friendly interface to visualize, browse, and annotate multi-camera video content. The tool is described in details in deliverable D3.2 [2]. In short, the interface allows for fluent multi-view content browsing, so as to manually defining events and positions of salient objects along the time. The notion of fluent browsing is defined through the user manual provided in Annex 1 of deliverable D3.2 [2]. Next to annotation menus (based on the XML format defined in D2.1 [3]), it involves the exploitation of a preview for all camera views, easy camera selection for the main view, temporal scrolling, and zooming capabilities. The events of interest correspond to clock-events, as defined in Section 5 of deliverable D2.1 [3]. Salient objects correspond to basket-ball players and ball, and are defined based on a bounding box. From an implementation point of view, the design of the interface relies on the WxWidgets graphical toolkit and ffmpeg libraries considered by the common testbed.

### ***Generation of planar images from omnivision***

The test-bed functionalities are being augmented to support omnivision. A generic software is incrementally developed to generate planar images from omnidirectional images, and a first version is currently available.

As an example, Figure 4 displays an image of a basket ball court taken by a fisheye lens during the first acquisition campaign, and Figure 5 displays a planar image generated from the omnidirectional view.

The initial version of the software takes in input a set of five parameters: the direction of view, defined by two angles standing for azimuth and elevation in a world reference frame, the angle-of-view of the virtual pan-tilt-zoom camera, the resolution, and the ratio between horizontal and vertical resolutions of the planar image.

This software has been developed as Matlab routines, and is being incorporated in the test-bed c++ framework.

It will produce high resolution planar images as an output.



Figure 4: Original picture from camera3



Figure 5: Generated planar image

## 4 Conclusions

---

The current prototype of the test-bed has been developed under a common Linux development platform and a set of common utility libraries for accessing frames, time objects, the display, etc. The platform provides also c++ classes for exchanging messages between APIDIS partners' modules and a shared set of options that are specific to a given sport event (dates, main directory of video files, etc.) The test-bed uses the wxWindow graphical toolkit that provides native Operating System look and feel. The prototype currently allows for browsing the recorded video streams. A complete set of annotation tools has been developed using the same environment and an initial version of the module for generating planar images from omnivision has been delivered.

## 5 References

---

- [1] APIDIS deliverable. D3.1 Deployment of the acquisition and storage system.
- [2] APIDIS deliverable. D3.2 Raw data content provisioning.
- [3] APIDIS deliverable. D2.1 End-user requirements and architecture definition.