Wizard User Manual

A simple presentation engine for java based interactive digital tv.

Document release	00.04.00
Date	03/12/10
License	BY NC SA

Ut in pauca conferam..

Changelog:

Date	Ver:	Author	Description
28/10/10	00.01.00	Andrea Venturi	Draft from template
19/11/10	00.02.00	Thomas Pavani	First Complete Draft
02/12/10	00.03.00	Thomas Pavani	Better figures, added Static argument to the graphic tag
03/12/10	00.04.00	Andrea Venturi	Minor text revision before release

Table of contents

General Index

Introduction

What's digital tv after all?

Interactive tv is a long time mantra who many are presenting as the “next big digital thing” after digital hot water and digital sliced bread, anyway we are still awaiting it since 2004 or so and we are left with thirty year old teletext..

We have seen many trials and errors along the road, and we are not going to fill too many pages talking about them.

To make a long story short, we believe tv broadcasters need to move to graphic teletext, or something like this for the people to begin appreciate their bloody-expensive wall-sized new tv set; Wizard is a shot at this target.

As we are in Italy where java based tv set are broadly available, it's a very slim XLET application featuring a simple yet powerful presentation engine.

Of course, this is not at all the first presentation engine for MHP STB, but we believe can stand up for some features embedded, open source, lightweight, royalty free and well documented. This doc stands as last point proof.

Introduction to Avalpa

Avalpa is an Italian company working in the digital television market since a long time with a quite broad range of products and services for broadcaster, great skills, creativity and an open mind attitude. Avalpa http://www.avalpa.com/ is the main developer of OpenCaster and wrote this manual too; Avalpa offering include also:

- Custom installations and mainteinance of solutions
- Integration with others DVB systems
- Training, support and consultancy
- Further development of features and customization.

Contacts: info@avalpa.com for general questions or ordering info.

Introduction to Wizard, a simple presentation engine XLET

Wizard is a Java XLET (see Oracle JavaTv for the concept behind) compatible also with TV set-top-box with Multimedia Home Platform on board.

Wizard is a presentation engine that interprets file of descriptive-language strings in order to display on TV screen appealing graphical content and services.

In our mind Wizard purpose is to:

•Demonstrate the feasibility of MHP XLETs for reliable and useful services, open sourced for everybody to borrow programming best practices.

•Show a technology powerful enough to show the real potential of MHP applications (against the more traditional approach of presentation engines)

•Give away a tool that practically *enable rapid prototyping of* simple and advanced interactive tv services in a cost effective and efficient way.

Use cases: some examples

There many useful ways for effective usage of this java application.

Wizard can be used as a MHP stack / MHP STB testing tool .

Wizard can be used for the creation of Simple application for local/residential/hotel television DVB network with MHP enabled terminals.

Wizard can be used to refresh some old-fashioned services like teletext, easily adding graphical and multimedia elements to traditional text.

Wizard can be used as a presentation tool for a smartly-updatable digital-signage screen connected at a DVB-T/S/C network.

Intended audience of the doc

This documentation is written for many different potential viewers.

At the top, we can see people working on creating new ways to release content already digital (like web sites and cdrom) for the tv mass market. We believe they need to understand a bit of the technology behind interactive tv, and the usability principles we believe are key for user adoption, this Wizard technology we are releasing freely can help them become proficient and productive quickly.

The next group is technical people, like analyst, programmers or quality assurance people, who need to use, test, extend and support interactive tv technologies so need to understand the overview and the capability of Wizard.

Last we see traditional tv technicians who can easily check what they do need to put in place to enable interactive tv services into their digital headend.

Of course, all the people interested in Wizard as a technology for educational purposes, schools or self-student, are welcomed to read and criticize where they do believe it should be fixed or improved.

Technical overview

Wizard is an application (more precisely a XLET) written in JAVA language, in accordance to the Multimedia Home Platform (MHP) standard.

It's been designed to be the smallest codebase for ensure practical feasilibility also on first generation underpowered MHP STB. Anyway there are present also mechanism for extensibility and added business logic.

The basic footprint is less the 100KB of java classes (that can be easily reduced using standard carousel server compression technologies).

The basic concept of the engine is that the service is decomposed in a number of “pages”.

Each page is described in with a number of elements of different kind: text, images audio and so on.

The file format for content description is based on standard **Java Properties Files** as this is already present in the standard environment and doesn't need external XML processors and it's way more readable the XML files. Wizard tries to keep the fewer resources locked and access graphical scene using plain AWT primitives as other API are poorly supported in many stacks, at least this is our experience.

Many graphical items are kept, once they are loaded and decoded from the filsystems, into a cache for improving performances if they are used in next pages.

Everybody is welcomed here to help asking or developing new features, but keep in mind, the less the code the better.

Copyright and warranty

Wizard, the java application

Wizard, the java application, about the manual license

How to read the manual

This manual is provided AS IS, and nothing useful is guaranteed.

We have written this stuff in good faith and we believe there are not a lot of mistakes.

If you happen to catch one, please report to support@avalpa.com so we can fix it for you and for your mates in this crazy adventure on digital television revolution. Wouldn't you like that other people would make the same choice. Right?

It is strongly suggested to not skip any chapters of “Wizard User Manual” because we assume at every following chapter that the reader is confident with the content of the previous ones.

Every chapter also states where to find tutorial files.

```
The line written in this format are command line to be typed as is in the terminal
Example:
DtPlay sample.ts -t 110  -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

The following symbols will serve as follow:

Symbol	Description
🔍	Pay attention to this note, it should be a hint for a smarter usage of the software
⚠	Beware, you can get some undesirable effect if you don't understand instructions and follow them carefully
👤	Help! This part is not completely up to date or finished; maybe you can help us, would you like?

Then, in the paragraphs where there are practical exercises using materials to be found in the Wizard package, we list them in advance with such a graphical hint:

[Wizard/tutorial/file.xx]

What's an interactive application?

An interactive application is, in a few words, an application that responds to user activity. In the computer world almost all applications are interactive: from text processing to video games, from CAD programs to web browser, these applications respond to user inputs.

The TV world is a completely different scenario. The main use case of a TV set is passive: the user can only select from a set of video channels. The introduction of interactivity comes with teletext where the user can navigate through text page using the remote controller as input device.

The advent of digital television can bring a whole new spectrum of opportunities for expand the interactive-TV scenario. The integration of more and more powerful hardware within TV and set top box will permit to realize interactive applications more complex and appealing than plain teletext. In fact, digital TV sets contain an “engine” for running applications with graphical elements, animation, video contents, etc...

From another point of view, the spreading of interactive platforms as mass-user commodity should lead to the introduction of many new applications delivered by “digital” broadcaster as a new business opportunity. The great advantage of interactive TV is that all services are running in a controlled environment (unlike on the internet). Via DVB-T/S/C broadcast large audiences may be addressed without the need of scaling the server capacity or network connection.

The standardization of the interactive platform for digital TV led some year ago to **Multimedia Home Platform** (DVB-MHP). MHP is an openly defined middleware system using Java programming language and virtual machine. It enable the reception and execution of Java MHP applications called “XLETS”.

Wizard, the subject of this manual, is a full operative example of MHP XLET that also have the purpose to demonstrate the potential behind MHP XLETs.

What's a presentation engine?

A presentation engine is a software programs that interpret **code** and display information in a format closer to the natural language for the user .

Just like in the real word an interpreter translate for you an unknown or not-well-understood language, in the same manner a presentation engine translate a simple code language in a richer multimedia experience. One example for all - this is the way web browsers work: HTML code is interpreted and a page full of graphic and animation is presented to the user.

Wizard is a presentation engine that interprets file of descriptive-language strings in order to display on TV screen appealing graphical applications.

The market offering

There are many presentation engine on the market, lot of them focusing on “HTML rendering” with the hope to reuse the bulk of content present on the web service.

We believe it's not a paying strategy for many reasons. The main are:

1.there's too bloatiness and overhead on this kind of technology: in the digital TV scenario the communication bandwidth is a scarce and expensive commodity as well as MHP STB have limited computational power (in comparison to PC);

2.the user interface the web is based on is not good for browsing on TV: web browsing on pc screen is feasible as graphic resolution is very high, users are close to the screen and are “armed” with full QWERTY keyboard and mouse; TV sets have less resolution and are usually farer from users than a pc screen, finally TV set input device is a simple remote controller.

The search for an alternative solution to bloated proprietary HTML engines led us to create and release Wizard with its simpler description language. This is of course a a starting point where anyone is invited to join and improve.

Technicalities about compile and test Wizard and its examples

There are many things a technician can do about it. At least install properly the Wizard software on his environment.

We depict here three different lab scenarios, of course there are blends and eventually some more. Whatever scenario you forecast to use, Wizard is a Java application that must be compiled.

Build Wizard classes from source code:

In order to build Wizard classes it's necessary to install the JDK and get the MHP stub classes from www.mhp.org.

Then the command line for building is simply:

```
javac -source 1.2 -target 1.1 -classpath mhpstubs-src:source -d classes source/*.java
```

Test through uploading service to STB with the serial port

If you have a development MHP STB, sometime it is possible to upload the application (and the filesystem with data, text and images) over the serial port and so you can easily test and prototype the Wizard XLET. This is by far the cheapest way to test an application for real. We don't believe using PC emulators (fully fledged coming from renowned MHP providers or half baked like some present on the net) is useful enough as they usually hide where the problems are on the real production (user interface, you can't simulate a remote control with the PC keyboard; graphical results, font size, safe area, color selection)

Test with a carousel server and a realistic setup

In order to test Wizard within a realistic scenario, it is necessary to opportunely create a Transport Stream and broadcast it to the MHP STB with a DVB-T modulator. A TS is made of packets, a packet is the minimum unit that cannot be splitted, has a fixed length of 188 bytes and always starts with a synch byte (0x47).

Each table or elementary stream in a transport stream is identified by a 13-bit PID. A demultiplexer extracts elementary streams from the transport stream in part by looking for packets identified by the same PID. In most applications, time-division multiplexing will be used to decide how often a particular PID appears in the transport stream.

Transport stream brings also a logic representation of the carried content, mainly with the concept of programs. A single program is described by a Program Map Table (PMT) which has a unique PID, and the elementary streams associated with that program have PIDs listed in the PMT. For instance, a transport stream used in digital television might contain three programs, to represent three television “channels”. Suppose each channel consists of one video stream, one or two audio streams, and any necessary metadata. A receiver wishing to decode a particular "channel" merely has to decode the payloads of each PID associated with its program. It can discard the contents of all other PIDs.

If we want interactive decoders to be aware of applications within the transport stream, we need to insert the Application Information Table signalling.

In fact, the complete basic workflow at decoder side for receiving applications is:

1.the decoder tunes the channel

2.the decoder parses PAT and PMT

3.the decoder parses the AITs signalled by the PMT

4.the decoder presents to the user the application list got from the AITs (it can be necessary to press the "app" button on the remote control)

5.the user selects an application

6.the decoder loads the DSMCC carousel referenced by the application descriptor or connect to the remote http server

7.the decoder executes the application

Configure the AIT table

In order to set-up Wizard into an MHP carousel it is necessary the specify the following parameters from the AIT

1.Starting URL of the first page (if not specified it will look for "start" properties file on its base directory)

2.Waiting message to display when the engine loads itself (recommended)

3.Username for dial-up connection (optional)

4.Password for dial-up connection (optional)

5.Phone number to call for dial-up connection (optional).

How to setup a transport stream and configure the AIT table within the stream is out of the scope of this document, but you will find an exhaustive guide within the Avalpa OpenCaster manual (please refer to www.avalpa.com).

Test on PC

It could be possible to run Wizard on a personal computer working as a MHP-STB emulator with lots of limitations and constraints.

Please note that it's necessary to have **JDK** installed, check **pc-emu** directory for help and an example. This is an effort quite tough and if you need productive solutions, we discourage quite heavily as it's by far more time consuming then the other approach, anyway it could be interesting from the educational point of view.

Please note that you could be able to partially execute Wizard as it does rely mainly on Java classe present in the standard Java CDC profile like AWT for graphic interface.

In the emulation environment the following tags are not working:

Audio, Background, Text (Dynamic), Service, Video, Event

The dvb:// protocol is not working too

The remote control coloured key are replaced as follows:

Red key is F1, Green is F2, Yellow is F3 and Blue is F4

Use cases

Here there are the main use cases we planned for this software; there content for general use (weather info, news, sport, finance like classical teletext) and contextual to event on air (like info about movie or docu, simple games like trivia)

You are free to find new formulas useful and spread the voice so we can improve this chapter..

Graphic televideo

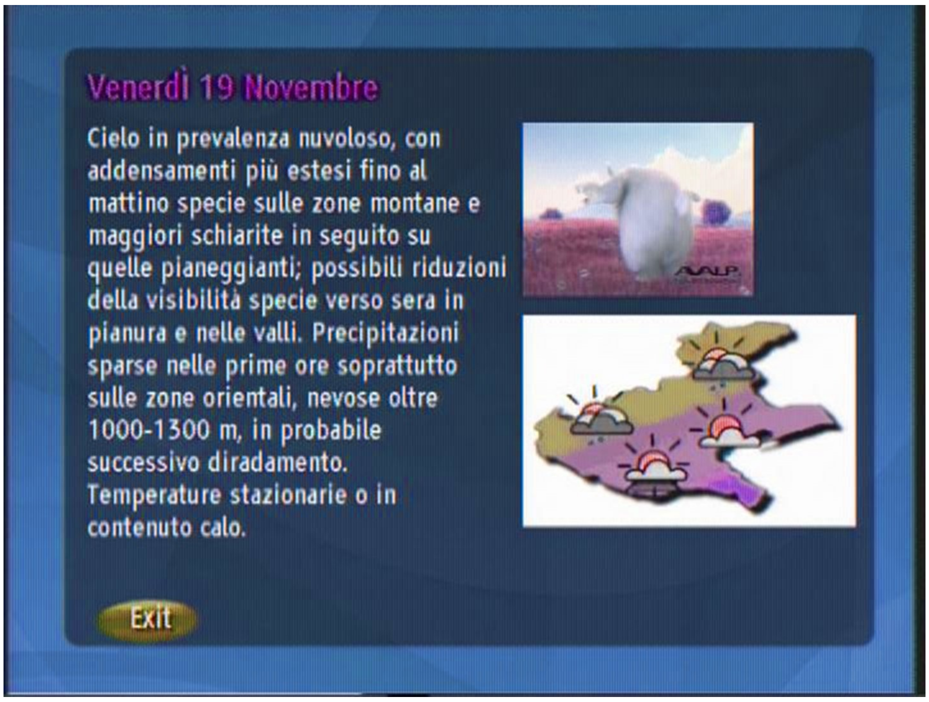
Wizard can be transformed in an teletext-like application, but with more pleasant look and feel. Introduction of image and graphical object will be easily managed. At the same time the teletext goto-page access can be integrated with some kind of word-search intelligence.

Of course for a production environment you'll need to put in place a “**content management system**” (CMS) on the backend so you could maintain easily a large set of pages. Avalpa is of course interested to do this next step as a consultancy project, eventually adapting some already available solution or designing a new one. This task could be also made by third parties and we would be happy to make a link here if it's freely and openly available.

Weather Info

It is possible to collect and manage weather information with a server side application, and, with the same application, to conform this info to the Wizard language. In this way Wizard can be re-designed as a Weather Info application. Here's a screenshot of Wizard displaying weather info for an Italian region.

In this example Wizard presents some page with textual and graphical info in slow rotation, leaving the video resized in the top-right angle of the screen.



Venerdì 19 Novembre

Cielo in prevalenza nuvoloso, con addensamenti più estesi fino al mattino specie sulle zone montane e maggiori schiarite in seguito su quelle pianeggianti; possibili riduzioni della visibilità specie verso sera in pianura e nelle valli. Precipitazioni sparse nelle prime ore soprattutto sulle zone orientali, nevose oltre 1000-1300 m, in probabile successivo diradamento. Temperature stazionarie o in contenuto calo.

Figure 1: Wizard transformed in a Weather Info Application

Stream event Information

Wizard can be used to presents additional information regarding a particular streaming content. During a movie or documentary, synchronously to specific scene, Wizard will pop-up some kind of info text or simply a coloured dot that means : "*Press the coloured button for more info!*". In the latter case Wizard will also manage the key pressure and the display of the opportune informative text. "Stream event" is quite an advanced feature, as you need to properly synchronize data with audio video presentation, this could be easily done with batch processing using OpenCaster tools.

Proposes Examples for Wizard

Here we speak about a number of test and examples that can give a proper idea on the technology features.

Basic usage

The basic usage of Wizard is simple: printing static info (text and images) on the screen, react to remote control key pressure, temporize actions, resize video, play audio. But the “magic” behind this basic usage is that you do not need to write a single line of code. In fact, for these basic operations, the Wizard descriptive meta-language is sufficient.

Following this concept, it is very simple to traduce human language instructions such as

•*Display that jpg image in position x,y*

•*Resize video to one-quarter*

•*When the red key is pressed go to page 2 (in the same “sample” file)*

in the tag-syntax of Wizard, the previous sentences become something like these lines (present in the starting file “sample”) :

```
1.Graphic.1 =True, x, y, file:///image.jpg
1.Video = 0,0,180,144
1.RedKey = file:///sample, 2
```

Advanced usage

There are many more advanced usage of Wizard, here we show something. In respect to basic usage the following actions require a slightly more complex tag protocol for Wizard.

Timed presentations of multiple pages

Tv is a dynamic media presented to people for a “lean back” experience, so also interactive tv services should present an automatic mode to present informations without user intervention; Wizard can realize this kind of experience with the **timeout feature**: after a programmed amount of time, the application will move from one page to another one, refreshing the screen for a dynamic effect. User of course will still have the choice to move to a different page when available.

Use the return channel (for on demand contents, profiling and transactions)

As you know, we cannot send OTA too much information as broadcast bandwidth is expensive and should be shared among all the potential viewer..

So MHP interactive tv has a trick to give everything to everyone (like in any good magic sketch), using the internet backbone!! (that was an easy answer..)

The large part of interactive STB has “return channel” (lowest common is a PSTN modem, highest an ethernet port). With such a link we can not only send request but also receive informations back!

Of course the PSTN is old, slow, and eventually expensive (premium rate numbers and so on..) but it's everywhere. On the other hand, the ethernet is (often) flat rate, high bandwidth and always available.. That's good for apps download and, in newer box, also for video on demand too.

Stream Events Management

Wizard can search within the TS stream for particular events called Stream Event. It is possible to instruct Wizard to take an action when the stream event is detected.

This feature permits to create applications that can take specific action when the broadcaster put the stream event on air, so it is possible to synchronize a running application to a video program on-going.

Manage input text from user

Interactive-TV means applications that respond to the user input. Usually people should move with just one press, but it is possible to go beyond the direct response to a single-key pressure. Wizard can manage more complex situations and collect text or numeric input typed with use of multiple keys; that is; user can use the remote control keys to write some text.

This feature opens the way for applications that take decision based on the text typed by the user: for example it is possible to design an application that collects (eventually through internet connection) and presents info about the subject typed by the user.

Offloading to external Business logic

There's also an useful way to extend the capability of Wizard, you can plugin external class and have them executed for special purposes.

Pay attention, **this way the STB could execute untrusted code**.

The description language of Wizard

Wizard is an application that can re-designed in many different ways. In order to tell exactly what Wizard have to do, a descriptive language has been developed.

As explained before, the HTML is not well suited for TV-set scenario: HTML means too bloatiness and overhead as well as web is based interfaces are not good for browsing on tv.

Even XML descriptive representation means additional JAVA classes for proper parsing to be broadcast along with Wizard. This is not bandwidth efficient and bring also to more unreadable content description.

The solution we use is to base the descriptive language of Wizard on the Java Properties file.

What's a file properties

In JAVA as well as in MHP, *Properties* are configuration values managed as *key/value* *pairs*. In each pair, the key and value are both String values. The key identifies, and is used to retrieve, the value, much as a variable name is used to retrieve the variable's value. For example, an application capable of downloading files might use a property named "download.lastDirectory" to keep track of the directory used for the last download.

Properties file parsing is a library part of any JAVA implementation so it was chosen over XML and others equivalent representation to avoid broadcast useless XLET classes for parsing.

The main methods provided by the Properties class enable to:

- loading key/value pairs into a Properties object from a stream,
- retrieving a value from its key,
  - adding a key/value pair to the Properties list

The tags

Contents and tags are stored into broadcasted files. The Wizard pages are contained in one or more files. Wizard identifies each page into a file by an Integer ID, starting from 1. So each page is uniquely defined inside the service with the couple: “filename,ID”

The Wizard properties files group property entries into *pages*. The properties within a page group describe the elements and actions that Wizard manage at one time. When Wizard moves to a new page, the graphical elements and the event-action triggers are constructed following the property entries belonging to the new page group.

The properties keys within Wizard property file follow a particular syntax in order to define pages.

Here's an abstract example of Wizard properties file:

```
#Wizard abstract properties file


1.Tag = value

1.Tag = value

1.Tag = value


2.Tag = value

2.Tag = value

2.Tag = value
```

This properties file define two page, each with 3 properties. In fact, the “#.” at the beginning of each property key assign that property to page identified by the number “#” .

In that way keys are constructed following a simple syntax:

```
Page_number.Tag
```

for single-use tags, that is, tags that can be used only once per page (for example, is possible to associate only a YellowKey tag per page, because it's possible to associate only an action triggered by the yellow button pressure), whereas

```
Page_number.Tag.Index
```

is used for non-single use tags; in this case Index is a integer value used for discriminate elements with the same tag type (for example it is possible to create a page with more Text elements, each with different characteristics, numbering each entry with a different Index).

Before examining in detail the different types of Tag usable for Wizard properties file, lets take a look to the syntax used for URL. URL is used to address file and special commands.

Wizard URL syntax is defined as:

```
protocol://string?variable_1=value_variable_1&...&variable_n=value_variable_n
```

URL supports different protocols: file://, dvb://, http://, class:// and exit://.

•**file://** loads from the current file system, note that "file://sample?name=giovanni&surname=bianchi" is a valid file's name.

•**dvb://** is followed by the carousel's id number, eg: "dvb://1/dir/file" will refer to the file into the directory 'dir' in the carousel with carousel's id '1'

•**http://** is the usual network protocol

•**class://** is a special protocol (usable only with certain tags), can be used to load an external class file: class://external/sample?variable1=value1 will load the class external and query the class passing "sample?variable1=value1" as parameter

•**exit://** is a special protocol (usable only with certain tags) used for exit from Wizard

If an URL with protocol file:// or dvb:// with at least a parameter fails to be found, Wizard will search the URL without the parameters' value, eg: if

```
dvb://1/query?name=giovanni&surname=bianchi
```

cannot be found, Wizard will search

```
dvb://1/query?name=&surname=
```

The next queries, however, will still try to search first the URL with valued parameters, so

```
dvb://1/query?name=giovanni&surname=bianchi&age=27
```

will be looked for before

```
dvb://1/query?name=&surname=&age=
```

if "name" and "surname" were valued at least once.

Graphic Tag

Load and display jpeg, gif or png images[1]

Usage:

```
Page_number.Graphic.number = Static, pixelX, pixelY, Display, URL
```

If more Graphic elements are defined for a single page, the rendering order follows the .number order.

The size of the rendered image is defined by the size in pixel of the jpeg, gif or png

•Static can be true or false, if true the image is loaded and cached by Wizard in order to prevent unnecessary loading work (when Wizard change page does not need to reload image loaded for the previous page), if false the image is not cached and Wizard will always reload the image in page changes and it will also monitor the image file status in the carousel: if the image is changed and the carousel is updated, Wizard will reload the image as well as repaint it on screen if Display is true.

•pixelX, pixelY represent the rendering coordinate in pixel of the top-left pixel of the image; negative values are valid.

•Display can be true or false, if false the image is only loaded

•URL define the source file for the graphic element

[Wizard/tutorial/sample1]

[Wizard/tutorial/sample1d]

Example 1

```
        1.Graphic.1 = True, 0, 0, True, file:///photo.jpg
```

This entry tells Wizard to load and display (Display=True) in page number 1 the JPG file photo.jpg from the current file system (URL begin with file://). The image is rendered with the top-left pixel in position 0,0 (that is, the top-left corner of the screen). The image size is defined by the size in pixel of the file photo.jpg. The graphic element is static (Static=True) so the image is cached and is not refreshed on carousel updates.

Example 2

```
        2.Graphic.1 = True, 0, 0, True, file:///square.png
        2.Graphic.2 = True, 100, 100, False, file:///triangle.png
```

1 see Appendix A for more details on graphical elements on screen

The first entry tells Wizard to load and display (Display=True) in page number 2 the PNG file square.png from the current file system (URL begin with file://). The image is rendered with the top-left pixel in position 0,0 (that is, the top-left corner of the screen). The image size is defined by the size in pixel of the file square.png. .

The second entry tells Wizard to load but not display (Display=False) in page number 2 the PNG file triangle.png from the current file system (URL begin with file://). The image is rendered with the top-left pixel in position 100, 100 (100 pixel left and 100 pixel down from the top-left corner of the screen). The image size is defined by the size in pixel of the file triangle.png.

Both graphic elements are static (Static=True) so the images are cached and are not refreshed on carousel updates.

Example 3

```
1.Graphic.1 = False, 0, 0, True, file:///image.jpg
```

This entry tells Wizard to load and display (Display=True) in page number 1 the JPG file image.jpg from the current file system (URL begin with file://). The image is rendered with the top-left pixel in position 0,0 (that is, the top-left corner of the screen). The image size is defined by the size in pixel of the file image.jpg. The graphic element is dynamic (Static=True) so the image is not cached and is refreshed on carousel updates.

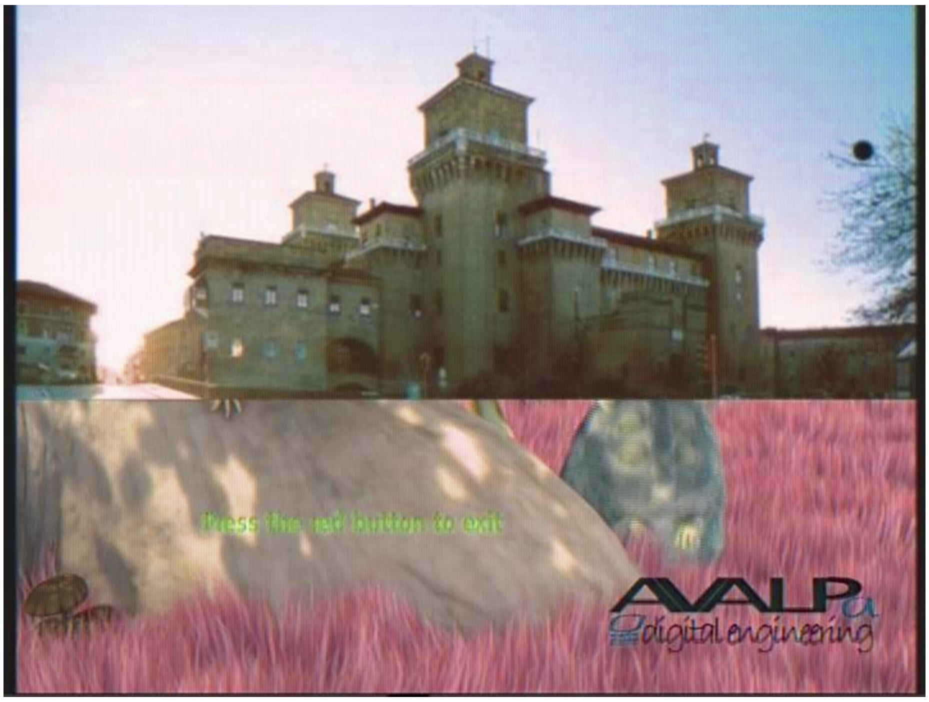


Figure 2: Example of image loaded and rendered over the video

Audio Tag

Mpeg2 audio is loaded and played at page change

Usage:

```
Page_number.Audio = URL, Playback
```

•URL define the source file for the audio element; only file:// and dvb:// protocol are supported

•Playback can be true or false, if false the audio is only loaded

[Wizard/tutorial/sample5]

Example 1

```
1.Audio = file:///gong.mp2, True
```

This entry tells Wizard in page number 1 to load the mp2 file gong.mp2 from the current file system (URL begin with file://). Wizard will play (Playback=True) the audio file when page change.

Example 2

```
2.Audio = file:///gong.mp2, False
```

This entry tells Wizard in page number 2 to load the mp2 file gong.mp2 from the current file system (URL begin with file://). Wizard will not play the audio file (Playback=False).

Background Tag

Mpeg2 video I-Frame is loaded and used as background[2]

Usage:

```
Page_number.Background = URL
```

•URL define the source file for the video element

[Wizard/tutorial/sample6]

Example 1

```
    1.Background = file:///intro.mpg
```

This entry tells Wizard to load and play the Mpeg2 file intro.mpg from the current file system (URL begin with file://) as background for page number 1

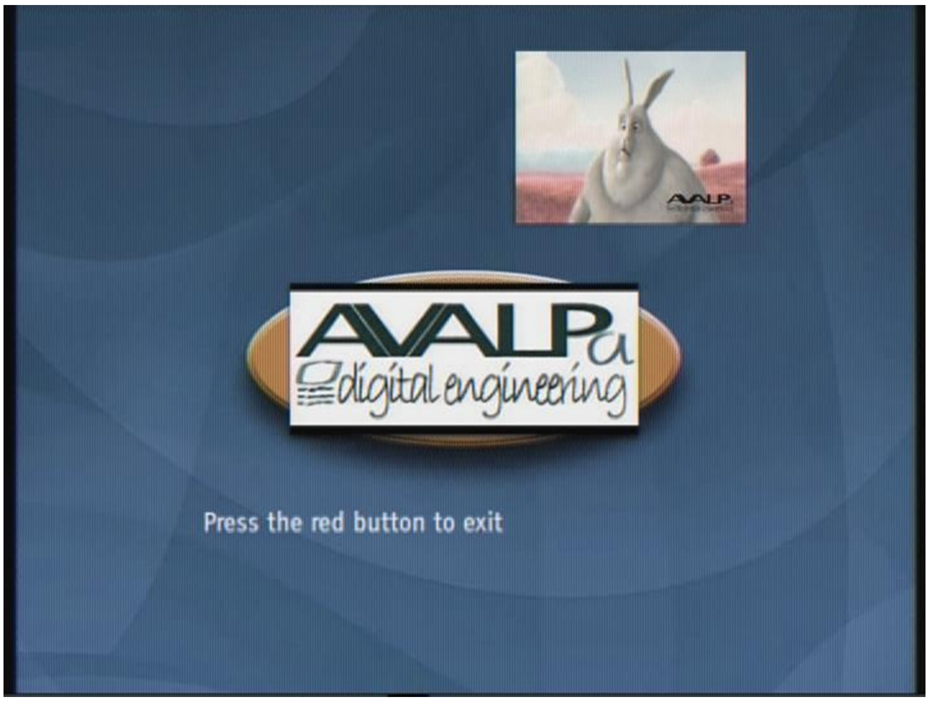


Figure 3: Example of background image (I-frame); please note that video is resized in order to visualize the background layer

2 see Appendix A for more details on background image and I-Frame

Text Tag

Print text on screen getting text from file

Usage 1 – Static Text:

```
Page_number.Text.number = Static(=True), pixelX, pixelY, Size, Color, Message
```

Usage 2 – Dynamic Text:

```
Page_number.Text.number = Static(=False), pixelX, pixelY, Size, Color, Display, URL
```

If more Text elements are defined for a single page, the rendering order follows the .number order.

The first argument determines if text is static (True) or dynamic (False). This choice changes the rest of arguments

In case of Dynamic text, the Text file should be unicode, render is updated if file's module version is changed

•pixelX, pixelY represent the rendering coordinate in pixel of the top-left pixel of the text; negative values are valid.

•Size define the size of characters.

•Color define the color of the text: the color must be described as hexadecimal string corresponding to the RGB color-space coordinates[3].

•Message (Static only) is the string to be rendered on the screen

•Display (Dynamic only) can be true or false, if false the text is only loaded

•URL (Dynamic only) define the source file for the text element

[Wizard/tutorial/sample2]

Example 1

```
1.Text.1= True, 150, 200, 24, 0x00FF00, Hello World!
```

This property describes a Static Text (first argument is True). This entry tells Wizard to render in page number 1 the string “Hello World!”. The text is rendered with the top-left pixel in position 150, 200 (150 pixel left and 200 pixel down from the top-left corner of the screen) with a size of 24 pixel and green color (0x00FF00 means R=0;G=255;B=0).

Example 2

3 see Appendix B for more details on color scheme coding

```
2.Text.1= True, 0, 0, 30, 0xFFFFFF, Press Red button to Exit
```

This property describes a Static Text (first argument is True). This entry tells Wizard to render in page number 2 the string “Press Red button to Exit”. The text is rendered with the top-left pixel in position 0,0 (that is, the top-left corner of the screen) with a size of 30 pixel and white color (0xFFFFFF means R=255;G=255;B=255)

Example 3

```
1.Text.1= False, 200, 100, 25, 0xFF0000, True, file:///info.txt
```

This property describes a Dynamic Text (first argument is False). This entry tells Wizard to load and display (Display=True) in page number 1 the text contained in file info.txt from the current file system (URL begin with file://). The text is rendered with the top-left pixel in position 200, 200 (200 pixel left and 100 pixel down from the top-left corner of the screen) with a size of 25 pixel and red color (0xFF0000 means R=255;G=0;B=0)

Example 4

```
2.Text.1= False, 0, 0, 30, 0x7F007F, True, file:///message.txt
```

This property describes a Dynamic Text (first argument is False). This entry tells Wizard to load and display (Display=True) in page number 2 the text contained in file message.txt from the current file system (URL begin with file://). The text is rendered with the top-left pixel in position 0,0 (that is, the top-left corner of the screen) with a size of 30 pixel and purple color.

Video Tag

Video is scaled and moved as specified

Usage:

```
Page_number.Video = pixelX, pixelY, Width, Height
```

•pixelX, pixelY represent the rendering coordinate in pixel of the top-left pixel of the video; negative values are valid (but see note 1).

•Width, Height represent the the new width and height in pixel for the video

Some MHP stacks do not support moving video in a position with negative coordinate

Some MHP stacks do not support resizing video to size under 180x144 pixel; further in some case resizing under this side is accepted but generates some cropping effects on the scaled video.

[Wizard/tutorial/sample3]

Example 1

```
1.Video = 0, 0, 320, 240
```

This entry tells Wizard to resize the video to 320x240 pixel and move it to position 0,0 (that is, top-left pixel of the video in the top-left corner of the screen) when in page number 1.

Example 2

```
2.Video = 80, 80, 180, 144
```

This entry tells Wizard to resize the video to 180x144 pixel and move it to position 80,80 (top-left pixel of the video 80 pixel left and 80 pixel down from the top-left corner of the screen) when in page number 2.

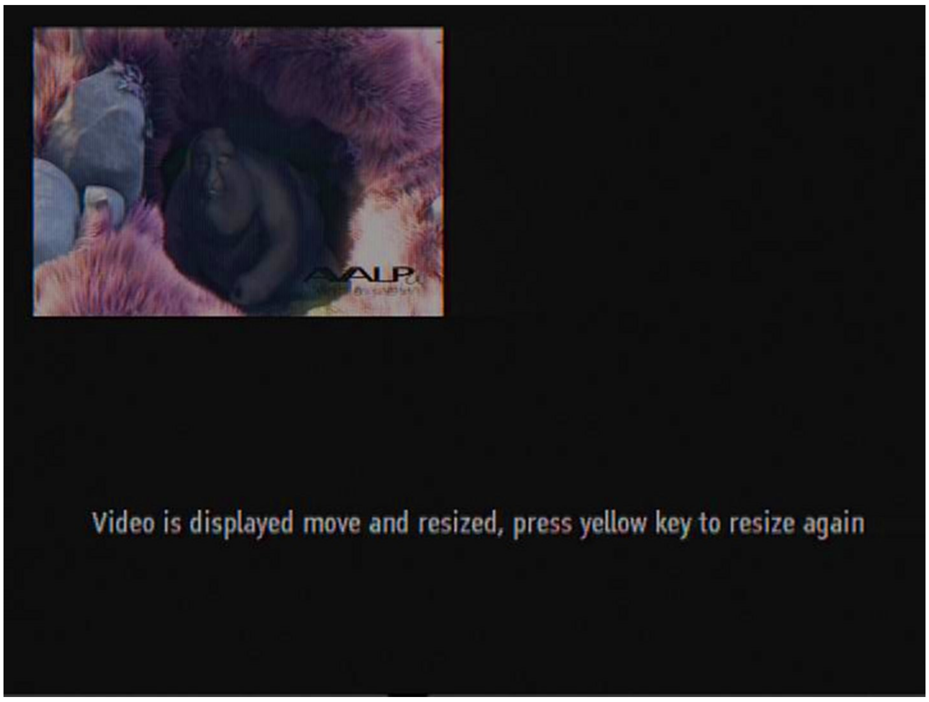

Video is displayed move and resized, press yellow key to resize again

Figure 4: Screenshot of an example of video resizing and moving

VideoCrop Tag

Video is scaled and moved as per the Video tag, but it also possible to select only a portion of the original video, enabling video cropping and zooming.

Usage:

```
Page_number.Video = pixelX, pixelY, Width, Height,
source_pixelX, source_pixelY, source_Width, source_Height
```

•pixelX, pixelY represent the rendering coordinate in pixel of the top-left pixel of the video; negative values are valid (but see note 1).

•Width, Height represent the the new width and height in pixel for the video

•source_pixelX, source_pixelY define the starting point (top-left) of the selection rectangle for the source video

•source_Width, source_Height define the width and height of the selection rectangle for the source video

If the parameters are set to: source_pixelX = 0, source_pixelY = 0, source_Width = 720 and source_Height= 576 the VideoCrop tag works exactly as the Video tag

Some MHP stacks may not support all the resizing, cropping and zooming combination enabled by this tag

[Wizard/tutorial/sample3crop]

Example 1

```
        1.VideoCrop = 80, 80, 320, 240, 0, 0, 720, 576
```

This entry tells Wizard to resize the source video (full-screen selected, starting from pixel 0,0 with width=720 and height=576) to 320x240 pixel and move it to position 80,80 (that is, top-left pixel of the video in the top-left corner of the screen) when in page number 1. This case produces the same result of 1.Video= 80, 80, 320, 240

Example 2

```
        2.VideoCrop = 100, 100, 180, 144, 0, 0, 180, 144
```

This entry selects the top-left quarter portion of the source video (starting from pixel 0,0 with width=720 and height=576), resize the video to 180x144 pixel (in effect no resize at all) and move it to position 100, 100 when in page number 2. The result is the cropping of top-left quarter of the screen, moved in position 100, 100

Example 3

```
        2.VideoCrop = 0, 0, 360, 576, 0, 0, 360, 576
```

This entry selects the left half portion of the source video (starting from pixel 0,0 with width=360 and height=576), and display the selected area without move or resize (same coordinate and size) when in page number 2. The result is right-half of the video not visible.

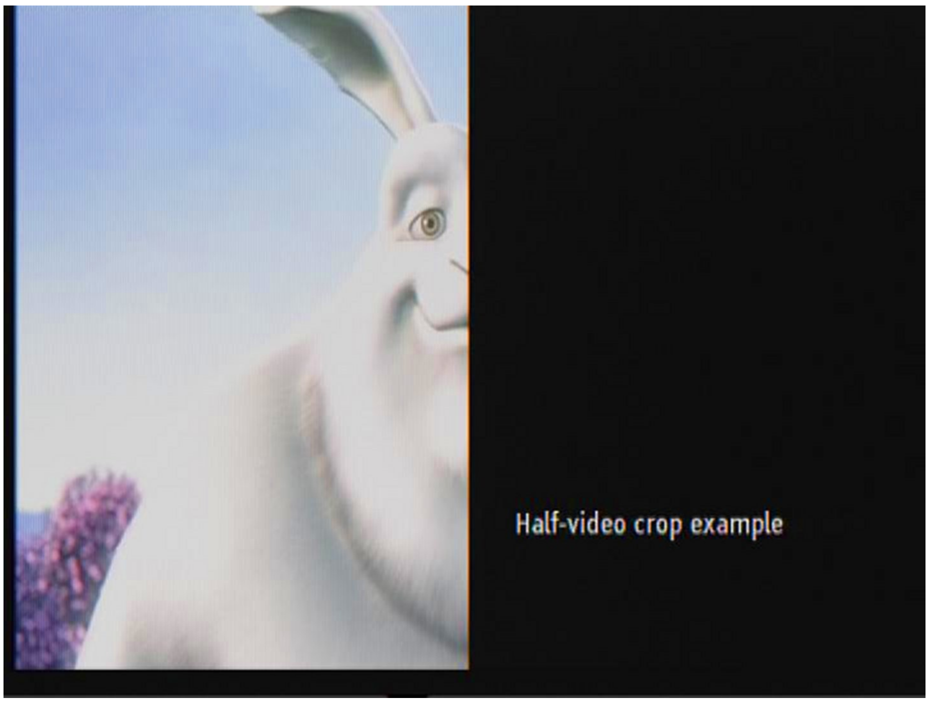


Figure 5: Screenshot of an example of video crop

Timer Tag

Set a timer to move at the specified page

Usage:

```
Page_number.Timer = seconds, URL, target_page_number
```

•seconds define the timer duration in seconds

•URL defines the property file describing the target page (exit:// is acceptable)

•target_page_number define the page to move to after timer elapsed

[Wizard/tutorial/sampletimer]

Example 1

```
1.Timer = 7, file:///sample2, 2
```

This entry tells Wizard to move from page number 1 (the property is associated to page 1) to page number 2 of the file sample2 from the current file system (URL begin with file://) after 7 seconds.

Example 2

```
3.Timer = 20, exit://, 1
```

This entry tells Wizard to move to page number 2 of the file sample2 from the current file system (URL begin with file://) after 7 seconds.

Service Tag

Changes service

Usage:

```
Page_number.Service = service_number
```

•Service number defines the service to move to; the syntax is

```
Dvb://network_id.transportstream_id.service_id
```

Where network_id, transportstream_id, service_id are all expressed as hexadecimal values (that is, from 0 to F).

Changing service automatically should terminate Wizard, as this is specified behaviour in the MHP 1.x life cycle, if not signaled on the new service or “bound” (clause 9.1.4.1 of ETSI TS 102812 ver.1.2.1 MHP 1.1)

[Wizard/tutorial/sampleService]

Example 1:

```
        2.Service = dvb://1.1.2
```

This entry tells Wizard in page 2 to switch to the service defined by network_id=1, transportstream_id=1, service_id=2.

Example 2:

```
        3.Service = dvb://1.3.b
```

This entry tells Wizard in page 3 to switch to the service defined by network_id=1, transportstream_id=3, service_id=11 (b in hexadecimal).

Event Tag

Subscribe to a stream event

Usage:

```
Page_number.Event = URL
```

•URL defines the stream event; Wizard subscribes to the stream event specified into the URL, and when the stream event is received changes page to the URL in the data part of the event.

[Wizard/tutorial/sample7]

Example 1:

```
        1-Event = file:///test.event
```

With this property Wizard in page 1 subscribes to the stream event test.event.

Key TAGs

RedKey

Moves at the specified page at the press of the red key

```
Page_number.RedKey = URL, target_page_number
```

•URL defines the property file describing the target page (both exit:// and class:// special URL are acceptable)

•target_page_number define the page to move to when the red key is pressed

Example 1

```
        1.RedKey  = file:///sample5, 2
```

This property tells Wizard to move from page number 1 (the property is associated to page 1) to page number 2 defined in the file sample5 from the current file system (URL begin with file://) when the red button is pressed.

Example 2

```
        2.RedKey  = exit://, 1
```

This property is defined for page number 2 and tells Wizard to exit from the application (URL is the special protocol exit://) when the red button is pressed.

YellowKey

Moves at the specified page at the press of the yellow key

```
Page_number.YellowKey = URL, target_page_number
```

•URL defines the property file describing the target page (both exit:// and class:// special URL are acceptable)

•target_page_number define the page to move to when the yellow key is pressed

Example 1

```
1-YellowKey  = http://remoteSample, 3
```

This property tellsWizard to move from page number 1 (the property is associated to page 1) to page number 3 defined in the file remoteSample when the yellow button is pressed. The file is remote, reachable via network (URL begin with http://)

BlueKey

Moves at the specified page at the press of the blue key

```
Page_number.BlueKey = URL, target_page_number
```

•URL defines the property file describing the target page (both exit:// and class:// special URL are acceptable)

•target_page_number define the page to move to when the blue key is pressed

Example 1

```
1.BlueKey  = file:///sample5, 2
```

This property tells Wizard to move from page number 1 (the property is associated to page 1) to page number 2 defined in the file sample5 from the current file system (URL begin with file://) when the blue button is pressed.

GreenKey

Moves at the specified page at the press of the green key

```
Page_number.GreenKey = URL, target_page_number
```

•URL defines the property file describing the target page (both exit:// and class:// special URL are acceptable)

•target_page_number define the page to move when the green key is pressed

Example 2

```
2.GreenKey  = exit://, 1
```

This property is defined for page number 2 and tells Wizard to exit from the application (URL is the special protocol exit://) when the green button is pressed.

Input Tag

Creates an input form; when 'OK' key is pressed Wizard moves to another page using typed text as variable.

It is recommended to use with tag in union with the PadHelp tag.

Usage:

```
Page_number.Input = pixelX, pixelY, width, ColorActive, ColorNotActive, onlyNumber,
                    VariableName, URL, target_page_number
```

•pixelX, pixelY represent the rendering coordinate in pixel of the top-left pixel of the input text; negative values are valid.

•Width defines the width of characters

•ColorActive defines the color of the currently typed text: the color must be described as hexadecimal string corresponding to the RGB color-space coordinates[4].

•ColorNotActive define the color of the already typed text: the color must be described as hexadecimal string corresponding to the RGB color-space coordinates[4].

•onlyNumber can be true or false, if true the input accept only number (and not letters)

•VariableName, URL the combination of this arguments and the typed text define the properties destination file to move to when the 'OK' key is pressed; the syntax is:

```
URL?Variable=typedText
```

•target_page_number define the page to move within the properties destination file

[Wizard/tutorial/sample8]

[Wizard/tutorial/sample9]

Example 1:

```
1.Input = 120, 180, 25, 0xFF0000, 0xFFFFFF, False, name, file://sample8, 1
```

This property tells Wizard to accept input from the user. The user can write text through use of the remote control. In this case it is possible to type both numbers and letters (onlyNumber parameter is False; please note that if the PadHelp is present it will display both numbers and letters). Typed text will be rendered at position 120, 180 pixel with size of 25 pixel. The active character will be red (0xFF0000 is red), whereas already typed text will became white (ColorNotActive parameter is 0xFFFFFF=white). Lets assume that the user typed “Mark”. When user press the 'OK' key, Wizard switch to page 1 of the file

```
sample8?name=Mark
```

in the current file system (URL begins with file://). If this file fails to be found, Wizard will

4 see Appendix B for more details on color scheme coding

search the URL without the name parameters value, that is:

```
sample8?name=
```

Example 2:

```
1.Input = 120, 180, 30, 0x0000FF, 0xFFFFFF, True, phone, class://sample9, 1
```

This property tells Wizard to accept input from the user. The user can write text through use of the remote control. In this case it is possible to type only numbers (onlyNumber parameter is True; please note that if the PadHelp is present it will display only numbers). Typed text will be rendered at position 120, 180 pixel with size of 30 pixel. The active character will be blue (0x0000FF is blue), whereas already typed text will became white (ColorNotActive parameter is 0xFFFFFF=white). Lets assume that the user typed the number 123456. When user press the 'OK' key, Wizard load and instance external class file sample9.class (URL begins with class://). Then Wizard will call the method getProperty of the external class passing the string “phone=123456” as argument, that is:

```
class9_instance.getProperty(“phone=123456”)
```

This call should return a new properties object that Wizard will display.

PadHelp

Shows the help-pad for text input at the specified location

The PadHelp must be used in conjunction with the Input tag; a PadHelp tag in a property file without a input tag in the same page is simply ignored.

Usage:

```
Page_number.PadHelp = pixelX, pixelY
```

•pixelX, pixelY represent the rendering coordinate in pixel of the top-left pixel of the input text; negative values are valid.

If the input tag in the same page is defined to accept only numbers, the PadHelp display only number, otherwise both numbers and letters are visible

[Wizard/tutorial/sample8]

[Wizard/tutorial/sample9]

Example:

```
1.PadHelp = 270, 200
```

This entry tells Wizard to display a help pad (see the following screen-shots) in position 270, 200.

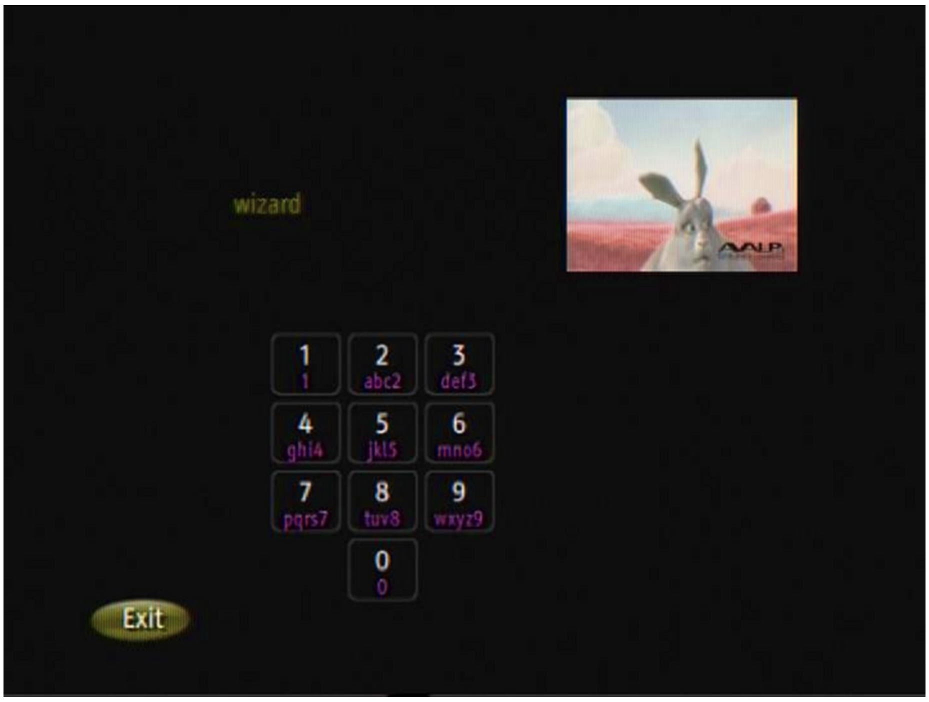


Figure 6: Screenshot of example of Pad Help graphic element - both numbers and letters are visible and typeable

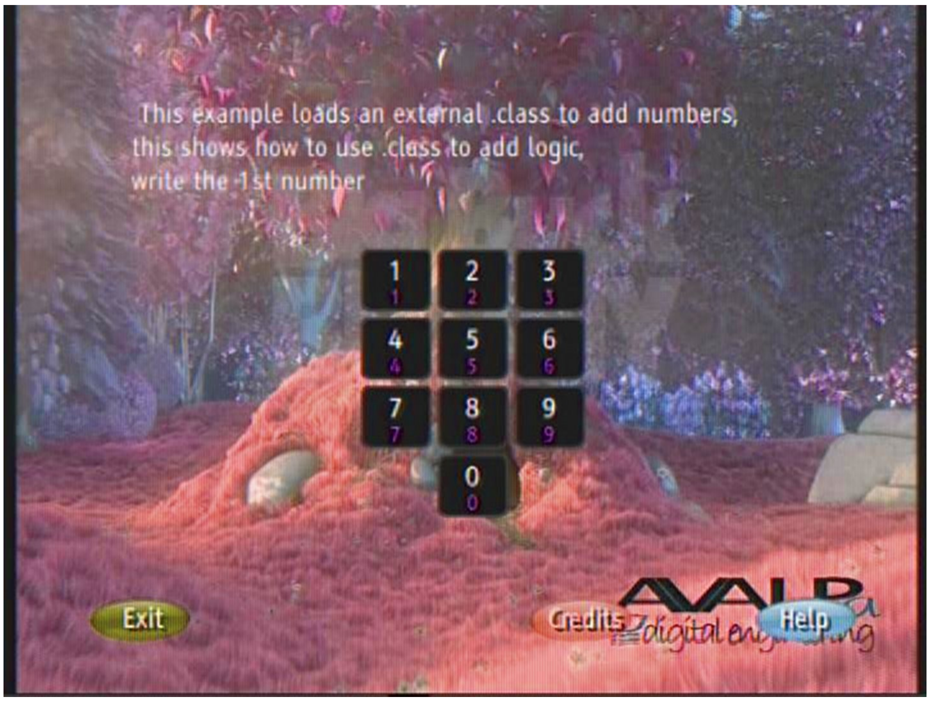

This example loads an external .class to add numbers,
this shows how to use .class to add logic,
write the 1st number

Figure 7: Example of help pad with only numbers

FailConnect Tag

This tag define the action to take in case of remote connection failure

Usage:

```
Page_number.FailConnect = URL, target_page_number
```

•URL defines the property file describing the target page (both exit:// and class:// special URL are acceptable)

•target_page_number define the page to move to when an remote connection failure event occurs

[Wizard/tutorial/sample4]

Example 1:

```
1.FailConnect = file:///sample8, 2
```

This property tells Wizard to move from page number 1 (the property is associated to page 1) to page number 2 defined in the file sample4 from the current file system (URL begin with file://) when a connection failure is detected.

Of course, there are many reasons to get a failed connection like missing cable, busy phone line, wrong number and so on. There's no way to get more details from the modem subsystem so the application designer should send to end user useful but generic error messages trying to hint to all these kind of problems.

Disconnect Tag

Close the connection if it is open

Usage:

```
Page_number.Disconnect = true
```

This property have a fixed syntax.

Example:

```
4.Disconnect = true
```

When Wizard load page number 4 disconnect any open dialup connection.

Full Example

Lets take a look to a full Example, where different tags are used to create a working application.

This example uses several file in order to manage user input and an external class. The file are

```
sample9
sample9?number1=
ExternalExample.class
```

The main file is sample9; it is a standard Wizard properties file:

sample9

```
# Input & class example

1.Text.1 = True, 100, 100, 25, 0xFFFFFF, This example loads an external .class to add
numbers,\nthis shows how to use .class to add logic,\nwrite the 1st number
1.Text.2 = True, 90, 523, 25, 0x000000, Exit
1.Text.3 = True, 87, 520, 25, 0xFFFFFF, Exit
1.Graphic.1 = True, 52, 494, True, file:///redb.jpg
1.Text.4 = True, 422, 523, 25, 0x000000, Credits
1.Text.5 = True, 419, 520, 25, 0xFFFFFF, Credits
1.Graphic.2 = True, 396, 494, True, file:///yellowb.jpg
1.Text.6 = True, 602, 523, 25, 0x000000, Help
1.Text.7 = True, 599, 520, 25, 0xFFFFFF, Help
1.Graphic.3 = True, 571, 494, True, file:///blueb.jpg
1.Input = 170, 150, 20, 0xFFFFFF, 0x000000, True, number1, file:///sample9, 1
1.PadHelp = 270, 200
1.RedKey = exit://, 1
1.BlueKey = file:///sample9, 2
1.YellowKey = file:///sample9, 3

2.Text.1 = True, 100, 100, 25, 0xFFFFFF, How to input text: Remote controller: -0-9 to input
number and letters\n-left arrow to delete\n-right arrow to space\n-ok to send text\nThis
example loads an external .class to add numbers,\nthis shows how to use .class to add
logic,\nwrite the 1st number
2.Text.2 = True, 90, 523, 25, 0x000000, Exit
2.Text.3 = True, 87, 520, 25, 0xFFFFFF, Exit
2.Graphic.1 = True, 52, 494, True, file:///redb.jpg
2.Text.4 = True, 422, 523, 25, 0x000000, Credits
2.Text.5 = True, 419, 520, 25, 0xFFFFFF, Credits
2.Graphic.2 = True, 396, 494, True, file:///yellowb.jpg
2.Text.6 = True, 602, 523, 25, 0x000000, Help
2.Text.7 = True, 599, 520, 25, 0xFFFFFF, Help
2.Graphic.3 = True, 571, 494, True, file:///blueb.jpg
2.Input = 170, 150, 20, 0xFFFFFF, 0x000000, True, number2, file:///sample9, 1
2.PadHelp = 270, 200
2.RedKey = exit://, 1
2.BlueKey = file:///sample9, 2
2.YellowKey = file:///sample9, 3

3.Text.1= True, 100, 150, 31, 0xFFFFFF, This software was realized by Avalpa.\nWizard is
GPL, actual version is 1.0.\nCheck www.avalpa.com
3.Text.2= True, 90, 523, 25, 0x000000, Exit
3.Text.3= True, 87, 520, 25, 0xFFFFFF, Exit
3.Graphic.1= True, 52, 494, True, file:///redb.jpg
3.Text.4= True, 250, 523, 25, 0x000000, Input
3.Text.5= True, 247, 520, 25, 0xFFFFFF, Input
3.Graphic.2= True, 224, 494, True, file:///greenb.jpg
3.Text.6= True, 422, 523, 25, 0x000000, Credits
3.Text.7= True, 419, 520, 25, 0xFFFFFF, Credits
3.Graphic.3= True, 396, 494, True, file:///yellowb.jpg
3.Text.8= True, 602, 523, 25, 0x000000, Help
3.Text.9= True, 599, 520, 25, 0xFFFFFF, Help
```

```
3.Graphic.4= True, 571, 494, True, file:///blueb.jpg
3.Text.10= True, 90, 523, 25, 0x000000, Exit
3.Text.10= True, 87, 520, 25, 0xFFFFFF, Exit
3.RedKey= exit://, 1
3.BlueKey = file:///sample9, 2
3.YellowKey = file:///sample9, 3
3.GreenKey = file:///sample9, 1
```

Sample9 is structured into 3 pages. In page 1 Wizard display on screen some text, an input form with help pad and some graphical reminder for the button action

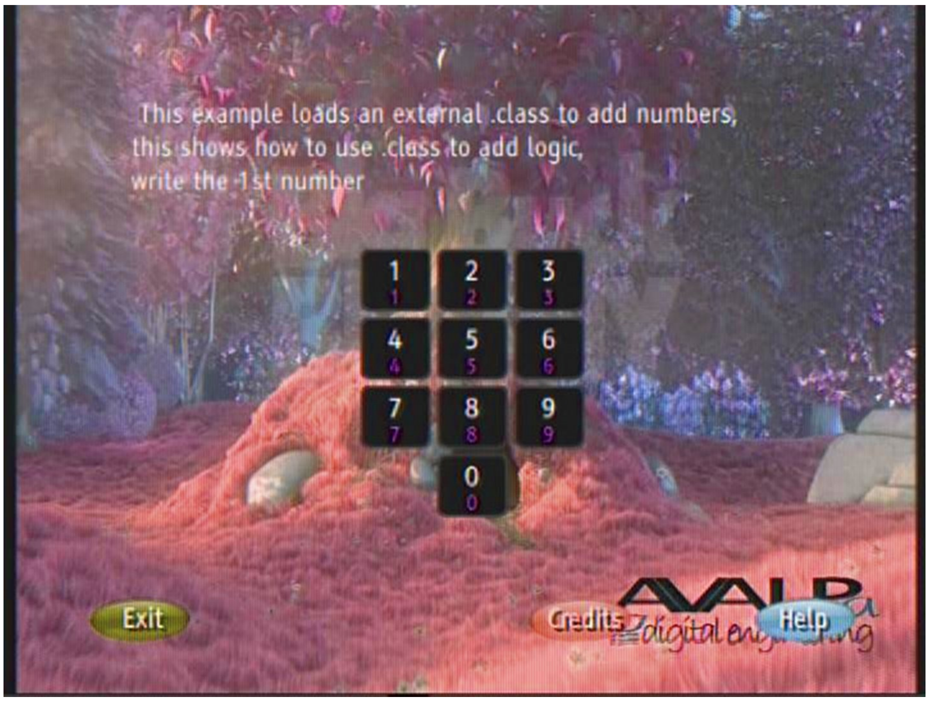

This example loads an external .class to add numbers,
this shows how to use .class to add logic,
write the 1st number

Figure 8: Sample9 page 1 screenshot

The entry 1.Text.1 draws white text , 1.Text.2 and 1.Text.3 draw the “Exit” label with a shadow effect (please note that both 1.Text.2 and 1.Text.3 draw the “Exit” label, but 1.Text.2 is slightly moved in respect to 1.Text.3 and is black colored).

The 1.Graphic.1 loads and displays a red icon under the “Exit” label in order to suggest to the user: ”Press the red button for Exit”.

The other labels and button are managed in a similar way. They are static (Static=True) graphical elements, so they are loaded and cached in order to prevent useless loading time as they appear in each page group of the properties file.

The entry 1.Input enables the user to type a number (in the 1.Input entry the onlyNumber argument is true), whereas the 1.PadHelp entry gives to the user a graphical guidance.

Finally the entries 1.RedKey, 1.BlueKey, 1.YellowKey, assign the action to take when the respective key is pressed: the red key pressure will exit Wizard, whereas the blue and yellow key will move Wizard to page 2 and page 3 respectively.

Page 2 is the help page. It is almost equal to page 1. The intro text becomes an explanatory text, and the blue “help” button is replaced by the green “input” button (on green button pressure Wizard will return to page 1); the rest is the same.

Page 3 is a credits page: there is no input and help pad elements, only a commentary text.

When Wizard is in page 1 or page 2 (the Input entry are equal), users can type a number using the remote control keys. Typing is terminate by the “OK” pressure. Lets assume the user typed the number 65. At the pressure of the “OK” key, Wizard will search for the file

```
sample9?number1=65
```

In the local file system. This is the combination of the URL argument (file://sample9) of the input entry, the variable's name (number1) and the typed text (65). Because this file does not exist, so Wizard will search for

```
sample9?number1=
```

The same file name without the argument value. This file exist and so Wizard load page 1 (as per the value indicated in the Input entry).

Here's *sample9?number1=* content:

```
1.Text.1= True, 100, 100, 25, 0xFFFFFF, ok, now write number 2
1.Input = 170, 150, 20, 0xFFFFFF, 0x000000, True, number2, class://ExternalExample/sample9?
number1=, 1
1.PadHelp= 270, 200
```

This page waits for another input from the user, another number because the input onlyNumber argument is true. Let's assume that the user typed number 78. When the “OK” key in pressed Wizard will load the external class ExternalExample.class as indicated in the URL. Then it will pass to the the getProperty of this class the string

```
number1=65&number2=78
```

This string is the result of concatenation of the first variable name and value with the second variable name and value.

Please note that the first variable value do not determines the file to be load after the first input action (it will be always the *sample9?number=* file) but it is stored and used in the external class.

The ExternalExample will parse the string, obtain the two number and sum them. The class will dynamically create a new properties object in which the sum result is inserted in a Text property. This properties object is returned to Wizard that will display page number 1 (as per the Input entry in *sample9?number1=* file).

WYSIWYG?

If you are looking for a graphical application for easy prototyping of pages and template, sorry, we do not have now.

You can of course help here with you ideas maybe pointing us on some package already available that can be easily hijacked to become a graphical configuration interface.

Deploying wizard in production.

Of course a real MHP application should be really dynamic and not show only static informations, otherwise the user would be quickly bored staring always in front of the same screen.

There's no time here to get in deep on dynamic servicesm but we just want to suggest to look at internet applications and try to imagine the same content to be poured by a tv set. A lot of the user interface has to be re-thought.

Help, community and further development.

Avalpa is committed on make Wizard the bleeding edge of the presentation engine for MHP (and not only) STB.

So we are ready to hear about issues and new features requested or better developed by third parties so they could be folded back into the GPL licensed package and become useful for everybody. That's the core value in open source development, after all.

Appendix A: MHP screen layering

In a MHP system graphic is displayed through the overlapping of three layers. From back to front, these are the background layer, the video layer, and the graphics layer.

The background layer cover all the screen (it will be visible in any area not covered by other two layers) and may display a single color, or alternatively a still image in the form of a single I-frame video.

Next is the video layer. It shows any video content. Full-screen video capability is mandatory for all MHP system, as well as the ability do resize video at quarter-screen resolution. More advanced receivers, however, can perform arbitrary scaling and positioning operations on the video.

The upper layer of the display stack is the graphical layer. This layer show any graphical object the MHP application is rendering. The graphical elements can be images loaded from file or text rendered in different colors or sizes. This layer grants far more freedom in respect to the background layer, however limitations exist anyway due to computational capability of MHP hardware systems.

I-Frame

The MPEG-2 standard specifies that the raw frames composing a video stream be compressed into three kinds of frames: intra-coded frames (**I-frame**), predictive-coded frames (**P-frames**), and bidirectionally-predictive-coded frames (**B-frames**).

An **I-frame** is a compressed version of a single uncompressed (raw) frame. It takes advantage of spatial redundancy and of the inability of the eye to detect certain changes in the image. Unlike P-frames and B-frames, I-frames do not depend on data in the preceding or the following frames.

Wizard accepts as valid background image single I-Frame video, that is, a video with a single frame compressed as I-Frame.

Two notes. Some receivers may use the hardware MPEG decoder for decoding I-frames and video drips, and for broadcasting video, and thus displaying images that use these formats may disrupt any video that is playing. This may be a short glitch while a frame is decoded, or it may last for the entire time the image is displayed, depending on the receiver.

The second thing we need to remember is that decoder format conversion will not be applied to any I-frames or video drips. These will be treated like full-screen images, no matter what aspect ratio or other format information is included in the images.

Appendix B: Colors scheme

Colors are displayed combining RED, GREEN, and BLUE lights.

Wizard accepts colors that are defined using a hexadecimal (hex) notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex 00). The highest value is 255 (hex FF).

Hex values are written as 3 double digit numbers, starting with a 0x sign:

0xRRGGBB

Color	Color HEX	Color RGB
⬛	0x000000	rgb(0,0,0)
🟥	0xFF0000	rgb(255,0,0)
🟩	0x00FF00	rgb(0,255,0)
🟦	0x0000FF	rgb(0,0,255)
🟨	0xFFFF00	rgb(255,255,0)
▨	0x00FFFF	rgb(0,255,255)
🟪	0xFF00FF	rgb(255,0,255)
▨	0xC0C0C0	rgb(192,192,192)
⬜	0xFFFFFF	rgb(255,255,255)

Appendix C: ExternalClass Interface

In order to add business logic to Wizard an ExternalClass management capability is provided.

External classes must implement the Interface provided in ExternalClass.java:

`ExternalClass.java`

```
import java.util.*

public interface ExternalClass{
        public Properties getProperty(CGIURL url)
}
```

Example: the following class implements ExternalClass and enables Wizard to sum two number and to display the result. The class assumes that the *url* object have the syntax

`number1=xxx&number2=yyy`

Where xxx and yyy are string representation of integer numbers.

With the call of `Property.setProperty` method, a Properties object is populated. Finally the Properties object is returned to Wizard for proper displaying.

ExternalExample.java

```
import java.util.*;
import java.io.*;

public class ExternalExample implements ExternalClass {

	/* This class has to stay at the carousel root, some mhp stacks have problems to load if
from others points */

	Properties Property;

	public Properties getProperty(CGIURL url) {

		int i = 0;
		int j = 0;

		String value = url.getValue("number1");
		if (value != null) {
			i = Integer.parseInt(value);
		}
		value = url.getValue("number2");
		if (value != null) {
			j = Integer.parseInt(value);
		}

		int k = i + j;

		Property = new Properties();
		Property.setProperty("1.Text.1", "True, 100, 100, 25, 0xFF0000, This is a .class
output example, result number is: " + k);
		return Property;
	}
```

Appendix D: Stream Events

This advanced topic chapter regards exactly how to signal interactive applications and Stream Event management. Stream events are signals sent on a particular PID to the interactive application and they are actually defined as all the other tables in a TS.

You can read more about it here:

http://www.interactivetvweb.org/tutorials/mhp/synchronization

The application, in order to receive the events, has to start executing and register itself to the events.

To register to event the application needs a reference object placed in the object carousel so it's necessary to create a Stream Event Object into the file system you want to broadcast.

How to create a **Stream Event Object** is out of the scope of this document, but you will find an exhaustive guide within the Avalpa OpenCaster manual (please refer to www.avalpa.com).