meztura

webKeyer

On-air Graphic System
based on Web Technologies

Index

Meztura WebKeyer, 3

What is Meztura WebKeyer?, 3

System Requirements, 3

Program operation:, 3

Correcly setting up live video input, 4

Displaying Flash and Silverlight content, 5

WebKeyer coding tips, 6

Meztura WebKeyer API - v1, 7

Introduction, 7

Communication system, 7

Authentication mechanism, 8

Meztura WebKeyer Protocol Specification, 8

Meztura WebKeyer

What is Meztura WebKeyer?

Meztura WebKeyer is a software that makes it possible to overlay HTML5/Javascript, Flash and Silverlight content on top of High Definition or Standard Definition video signals.

Think of it as a special Web Browser. In a normal Web Browser you type URL's where you want to navigate and the web pages are displayed in the browser window. The special thing about webKeyer is that it displays web content as the Playout of professional video cards and also that it preserves the transparency information (Alpha channel) of the rendered web pages allowing them to be overlaid on top of live video in real time.

System Requirements

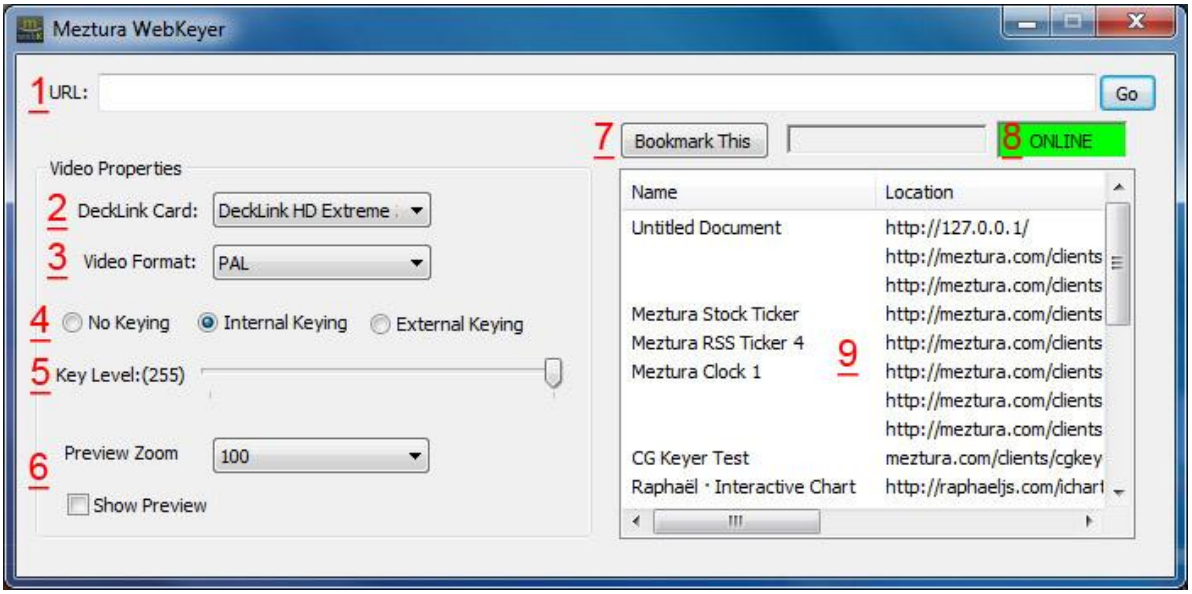
In order for webKeyer to work you need **Windows 7** (32 or 64 bits) and a video capture card. Currently we support the following video cards from the Black Magic Design manufacturer:

- **Decklink Extreme 3D** (allows us to overlay on top of HD video signals)
- **Decklink Studio** (allows us to overlay on top of SD video signals)

By using these cards we get a wide range of input/output connections (SDI, HDMI, Composite, Component,...) that makes them specially suitable for TV broadcasters and producers.

Program operation:

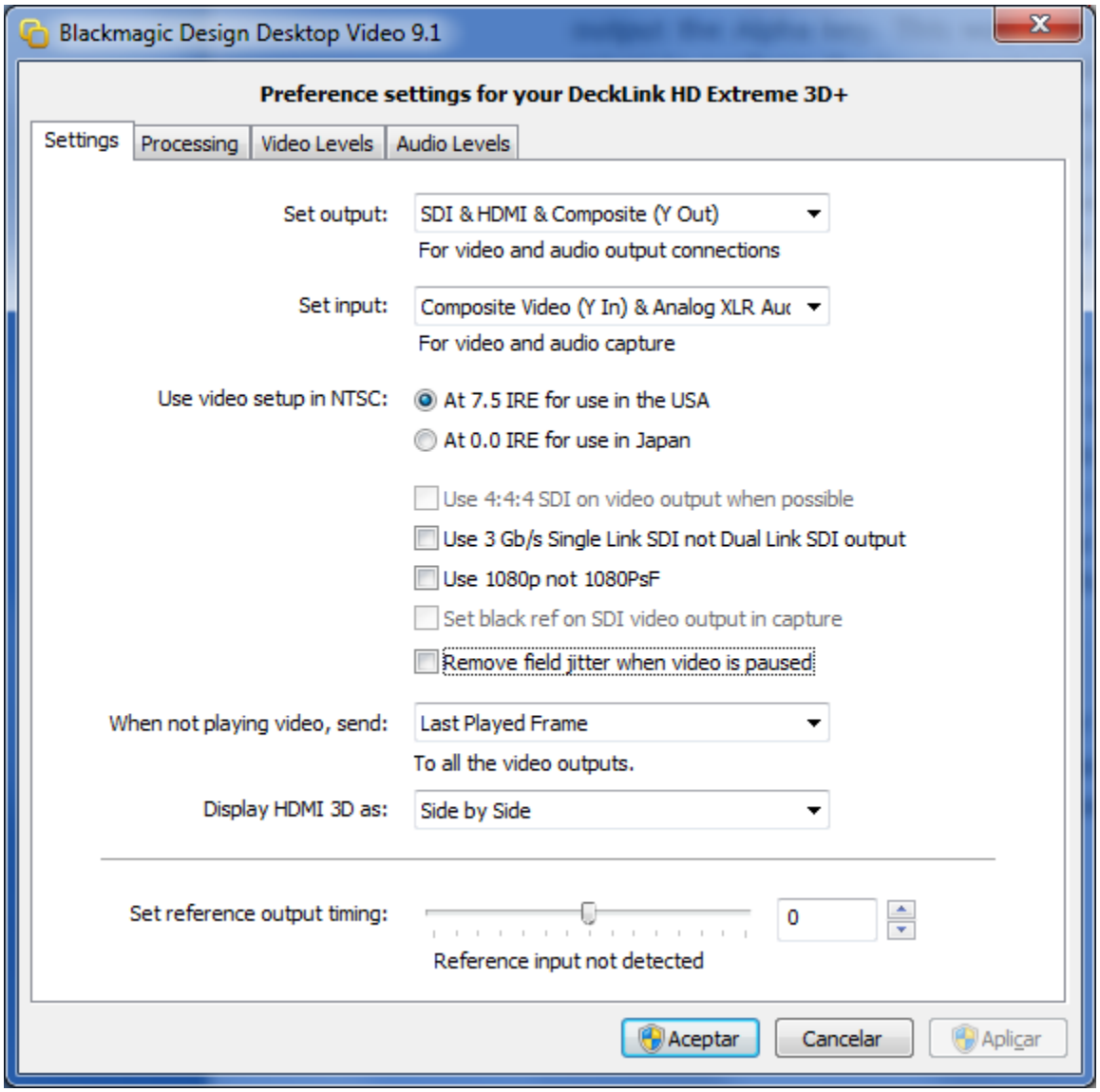
This is the main webKeyer window:



1. **URL field:** Introduce here any URL as you would do in a normal web browser for webKeyer to load it.
2. **Decklink card:** Select the Decklink card (if more than one is present in your system) that you would like webKeyer to use.
3. **Video format:** This control tells webKeyer rendering engine the video mode in which it has to output its contents. If you are using webKeyer to key content on top of live video signal this selector must match the video mode of the input live video.
4. **Key mode**:
    - **No keying:** webKeyer will output its content to the Decklink output but it won't do any alpha keying.
    - **Internal Keying:** webKeyer will key the web content on top of live video signal internally and send the result to the Decklink video output.
    - **External keying:** This mode only works for SDI. In this mode webKeyer will use one Decklink SDI output to output the Fill and the other one to output the Alpha key. This way you can use an external digital video mixer to perform the keying externally.
5. **Key level**: If you have selected internal key mode, this slider will set the key level of the content being overlaid on top of live video signal.
6. **Preview:** Displays a preview window in which you can see the content rendered by webKeyer. This preview window listens for mouse and keyboards events so that you can interact with the content being rendered as you would do in a normal web application.
7. **Bookmark button:** When you load a web page you can save it to your bookmarks. This way you don't have to type frequently used URL's.
8. **Online/Offline:** This button will Only work in Internal Key mode. It will activate and deactivate the content being keyed on top of video.
9. **Bookmarks:** All saved bookmarks will appear here. Double click on one to load it. Bookmarks can be deleted by pressing the "DEL" key.

Correcly setting up live video input

Prior to using webKeyer to key web content over live video you should check if you Decklink card configuration is correct. Go to Black Magic Design control panel and make sure you have selected proper video input and settings.



It is also advisable that you deactivate the "Remove field jitter when video is paused" to get better quality when you are working with interlaced SD video modes like PAL or NTSC.

Displaying Flash and Silverlight content

Meztura webKeyer uses the NPAPI plugins that are installed on the system. This means it uses the plugins that are designed for Firefox/Chrome.

In order to run Flash or Silverlight content in webKeyer, make sure these plugins are correctly installed and working in Firefox and Chrome or you can download it directly from Adobe's web site:

http://kb2.adobe.com/cps/142/tn_14266.html#main_Archived_versions

WebKeyer coding tips

There are not many. If it works in a normal web browser it will probably work in webKeyer.

There are although some things to take into account:

Web pages transparency (alpha channel): In order for transparency to work properly and be able to overlay HTML on top of video, background of your pages must be set to transparent or not defined (with no color) in the CSS style sheet.

```
body{ background-color:transparent; }
```

Scrollbars and margins: Probably you don't want to show the scroll bars of the webpage whenever some elements are overflown. You can easily remove them by setting the 'overflow' propiety of the 'body' element to 'hidden' in your CSS sheet.

By default all browsers setup a margin for the 'body' element. It would be convenient that you reset it to 0. This way you can make sure your content fits perfectly on screen:

```
body{ overflow: hidden; margin: 0px; padding: 0px; }
```

Meztura WebKeyer API - v1

Introduction

This section of the document describes the communication API exposed to developers and third party software by Meztura WebKeyer application.

Communication system

The communication with WebKeyer is based in the WebSockets protocol.

WebSocket is a web technology providing for bi-directional, full-duplex communications channels, over a single Transmission Control Protocol (TCP) socket. The WebSocket protocol has been standardized by the IETF as RFC 6455.

Meztura WebKeyer incorporates an internal WebSockets server that listens to incoming connections and establishes a bi-directional communication with a client. Once the connection is established a client can remotely control WebKeyer sending messages encoded in JSON format following the Meztura WebKeyer Protocol message scheme, as follows:

JSON message scheme that a client must send to WebKeyer:

{"type" : "request" , "id" : "commandName" , "data" : "additionalData"}

type: (String). This is a mandatory field. When a client sends a message to WebKeyer it must be set to "request".

id: (String). This is a mandatory field. Unique identificator for each possible commandName. See protocol specification bellow to see a list of all possible commands the WebKeyer accepts.

data: This is an optional field. The datatype of this field will vary depending on the need for each command. See protocol specification for more information.

JSON message scheme that a WebKeyer sends to clients:

{"type" : "response" , "id" : "commandName" ,"succeed" : bolean , "data" : "additionalData" , "error" : "errorMessage"}

type: (String). This is a mandatory field. When WebKeyer sends a message to a client it will be set to "response".

id: (String). This is a mandatory field. Unique identificator for each possible commandName. See protocol specification bellow to see a list of all possible commands the WebKeyer accepts.

succeed: (Bolean). This is a mandatory field. This field will return true if the operation succeed or false in case of failure.

data: This is an optional field. The datatype of this field will vary depending on the need for each command. See protocol specification for more information.

error: (String) This is an optional field. If the operation has not succeed, WebKeyer can use this field to inform the client about the error.

Authentication mechanism

For security reasons, a client connecting to WebKeyer must perform an authentication so that it can be checked that the client has control permissions.

User name, password and listening port in the server side can be configured with the "Meztura WebKeyer User Editor" tool bundled in the package.

In order to perform the authentication, the client must send the "**authClient**" message to the server prior to any other control message. Not doing so will force WebKeyer to respond with an authentication failure message.

See "**authClient**" bellow in the WebKeyer Protocol specification to get more info about this command.

Meztura WebKeyer Protocol Specification

What follow is a list of available commands when using the Meztura WebKeyer Protocol:

authClient:

This command will be used to authenticate a client and check if it has permissions to control WebKeyer. After connection is established with WebKeyer, **the client must run this command in order to be authenticated**. If authentication fails WebKeyer will send the client the authentication failure message. If any other message is sent to WebKeyer prior to authentication, WebKeyer will send the client the authentication failure message.

The md5username and md5password are the md5 hashes of both username and password:

Client:

{"type" : "request" , "id" : "authClient", "data" : {"username" : "md5username" , "password" : "md5password"} }

Server:

{"type" : "response" , "id" : "authClient" , "succeed" : true}

Server (in case of **authentication failure**):

{"type" : "response" , "id" : "authClient" , "succeed" : false, "error" : "Authentication error"}

getCurrentUrl:

This command will make WebKeyer to return a string containing the current URL loaded in WebKeyer.

Client:

{"type" : "request" , "id" : "getCurrentUrl"}

Server:

{"type" : "response", "id" : "getCurrentUrl" , "succeed" : true , "data" : "url"}

loadUrl:

This command will make WebKeyer to load a new URL by sending a 'url' string containing the new URL.

Client:

{"type" : "request" , "id" : "loadUrl" , "data" : "url"}

Server:

{"type" : "response", "id" : "loadUrl" , "succeed" : true , "data" : "loading"}

Server (when URL has been loaded):

{"type" : "response", "id" : "loadUrl" , "succeed" : true , "data" : "loaded"}

getCurrentCaptureDevice:

This command will return a string representing the currently selected video capture device in WebKeyer.

Client:

{"type" : "request" , "id" : "getCurrentCaptureDevice"}

Server:

{"type" : "response" , "id" : "getCaptureDeviceList" , "succeed" : true, "data" : "device1"}

getCaptureDeviceList:

This command will return an array of strings representing a list of all available video capture devices that WebKeyer is able to use.

Client:

{"type" : "request" , "id" : "getCaptureDeviceList"}

Server:

{"type" : "response" , "id" : "getCaptureDeviceList" , "succeed" : true, "data" : ["device1", "device2", "device3", ...]}

setCaptureDevice:

This command will select a video capture device from those available to WebKeyer.

Client:

{"type" : "request" , "id" : "setCaptureDevice", "data" : "device"}

Server:

{"type" : "response" , "id" : "setCaptureDevice" , "succeed" : true}

getCurrentVideoMode:

This command will return a string representing the currently selected video mode in WebKeyer.

Client:

{"type" : "request" , "id" : "getCurrentVideoMode"}

Server:

{"type" : "response" , "id" : "getCurrentVideoMode" , "succeed" : true, "data" : "mode1"}

getVideoModeList:

This command will return an array of strings representing a list of all available video modes that WebKeyer is able to use.

Client:

{"type" : "request" , "id" : "getVideoModeList"}

Server:

{"type" : "response" , "id" : "getVideoModeList" , "succeed" : true, "data" : ["mode1", "mode2", "mode3", ...]}

setVideoMode:

This command will select a video mode from those available to WebKeyer.

Client:

{"type" : "request" , "id" : "setVideoMode", "data" : "mode"}

Server:

{"type" : "response" , "id" : "setVideoMode" , "succeed" : true}

getKeyMode:

When working with video cards that support internal/external keying modes (see WebKeyer documentation), this command will get the current keying mode that WebKeyer is using.

There are three possible "keyMode" values: 'noKeying', 'internalKeying' and 'externalKeying'.

See WebKeyer documentation to get more information about these values.

Client:

{"type" : "request" , "id" : "getKeyMode"}

Server:

{"type" : "response" , "id" : "getKeyMode" , "succeed" : true, "data" : "keyMode"}

setKeyMode:

When working with video cards that support internal/external keying modes (see WebKeyer documentation), this command will set the corresponding keying mode.

There are three possible "keyMode" values: 'noKeying', 'internalKeying' and 'externalKeying'.

See WebKeyer documentation to get more information about these values.

Client:

{"type" : "request" , "id" : "setKeyMode", "data" : "keyMode"}

Server:

{"type" : "response" , "id" : "setKeyMode" , "succeed" : true}

getKeyLevel:

When working with video cards that support internal/external keying modes (see WebKeyer documentation), this command will get the current key level (opacity) of the web content rendered by WebKeyer on top of live video input. Server will return a 'keyLevel' value which is an interger between 0 and 255 being '0' fully transparent and '255' fully opaque.

Client:

{"type" : "request" , "id" : "getKeyLevel"}

Server:

{"type" : "response" , "id" : "getKeyLevel" , "succeed" : true, "data" : "keyLevel"}

setKeyLevel:

When working with video cards that support internal/external keying modes (see WebKeyer documentation), this command will set the key level (opacity) of the web content rendered by WebKeyer on top of live video input. 'keyLevel' is an interger value between 0 and 255 being '0' fully transparent and '255' fully opaque.

Client:

{"type" : "request" , "id" : "setKeyLevel", "data" : "keyLevel"}

Server:

{"type" : "response" , "id" : "setKeyLevel" , "succeed" : true}

getBookmarkList:

This command will retrieve an **array of objects** containing all the bookmarked URL's stored in WebKeyer. Each returned object will have the following structure:

{"id" : "bookmark_id", "url" : "bookmarkUrl"}

where bookmark_id is a string representing a unique numeric value for each bookmark stored in WebKeyer and bookmarkUrl is a string with the URL address of the bookmark.

Client:

{"type" : "request" , "id" : "getBookmarkList"}

Server:

{"type" : "response" , "id" : "getBookmarkList" , "succeed" : true, "data" : [{"id" : "bookmark_id1", "url" : "bookmarkUrl1"}, {"id" : "bookmark_id2", "url" : "bookmarkUrl2"}, {"id" : "bookmark_id3", "url" : "bookmarkUrl3"}, ...] }

saveBookmark:

This command will save a URL as a 'bookmark' in WebKeyer. 'Bookmark' is a string containing the URL to be saved.

Server will return a "data" value containing the following object:

{"id" : "bookmark_id", "url" : "bookmarkUrl"}

where bookmark_id is a string representing the unique numeric value for the stored bookmark and bookmarkUrl is a string with the URL address of the bookmark.

Client:

{"type" : "request" , "id" : "saveBookmark", "data" : "bookmark"}

Server:

{"type" : "response" , "id" : "saveBookmark" , "succeed" : true, "data" : {"id" : "bookmark_id", "url" : "bookmarkUrl"} }

deleteBookmark:

This command will delete a bookmark stored in WebKeyer internal database. 'bookmark_id' is a string representing the a unique numeric value for each bookmark stored in WebKeyer

Client:

{"type" : "request" , "id" : "deleteBookmark", "data" : "bookmark_id"}

Server:

{"type" : "response" , "id" : "deleteBookmark" , "succeed" : true , "data" : "bookmark_id"}

getAudioDeviceList:

This command will return an array of strings representing a list of all available output audio devices that WebKeyer is able to use.

Client:

{"type" : "request" , "id" : "getAudioDeviceList"}

Server:

{"type" : "response" , "id" : "getAudioDeviceList" , "succeed" : true, "data" : ["device1", "device2", "device3", ...]}

getCurrentAudioDevice:

This command will return a string representing the currently selected audio device in WebKeyer.

Client:

{"type" : "request" , "id" : "getCurrentAudioDevice"}

Server:

{"type" : "response" , "id" : "getCurrentAudioDevice" , "succeed" : true, "data" : "device1"}

setAudioDevice:

This command will select an output audio device from those available to WebKeyer.

Client:

{"type" : "request" , "id" : "setAudioDevice", "data" : "device"}

Server:

{"type" : "response" , "id" : "setAudioDevice" , "succeed" : true}

getLocalPreviewStatus:

This command will return a previewStatus **bolean value** being it false if the local preview window of WebKeyer **is not** being displayed and true if the local preview window is being displayed.

Client:

{"type" : "request" , "id" : "getLocalPreviewStatus"}

Server:

{"type" : "response" , "id" : "getLocalPreviewStatus" , "succeed" : true, "data" : "previewStatus"}

getLocalPreviewZoom:

This command will return a previewZoom value being it a string of '25', '50', '75', or '100' representing the zoom value of the local preview window in WebKeyer.

Client:

{"type" : "request" , "id" : "getLocalPreviewZoom"}

Server:

{"type" : "response" , "id" : "getLocalPreviewZoom" , "succeed" : true, "data" : "previewZoom"}

setLocalPreviewZoom:

This command will set the previewZoom value in WebKeyer. Possible values are '25', '50', '75', or '100' representing the zoom value of the local preview window in WebKeyer.

Client:

{"type" : "request" , "id" : "setLocalPreviewZoom", "data" : "previewZoom"}

Server:

{"type" : "response" , "id" : "setLocalPreviewZoom" , "succeed" : true}

toggleLocalPreviewStatus:

This command will toggle the local preview window status. If the local preview window is being displayed it will be hidden after executing this command. On the other hand, if the local preview window is hidden it will be displayed after executing this command

Client:

{"type" : "request" , "id" : "toggleLocalPreviewStatus"}

Server:

{"type" : "response" , "id" : "toggleLocalPreviewStatus" , "succeed" : true}

executeJavascript:

This command will remotely execute a 'javascriptStatement' in the web application loaded in WebKeyer. 'javascriptStatement' is a string containing the javascript statement to be executed.

Client:

{"type" : "request" , "id" : "executeJavascript", "data" : "javascriptStatement"}

Server:

{"type" : "response" , "id" : "executeJavascript" , "succeed" : true}

getPreviewImage:

This command will return an image of the current web application being rendered by WebKeyer

The client will send a data object with two parameters:

- "enconding": It will be a string representing the encoding format for binary data transport. Default will be **base64url**.
- "imageFormat": a string representing the image format. Default will be **PNG**.

Client:

{"type" : "request" , "id" : "getPreviewImage", "data" : {"enconding" : "encondingFormat" , "imageFormat" : "imageFormat"} }

Server:

{"type" : "response" , "id" : "getPreviewImage" , "succeed" : true, "data" : "imageData"}

webKeyerStatus:

This command will return full information about WebKeyer status

Client:

{"type" : "request" , "id" : "webKeyerStatus"}

Server:

{
"type" : "response",
"id" : "WebKeyerStatus" ,
"succeed" : true ,
"data" :
{
"currentStatus" : "OK",
"currentUrl" : "url",
"currentVideoCaptureDevice" : "captureDevice",
"currentVideoMode" : "videoMode",
"currentKeyingMode" : "keyingMode",
"currentKeyLevel" : "keyLevel",
"currentLocalPreviewZoom" : "previewZoom",
"currentLocalPreviewStatus" : "previewStatus",
"currentAudioDevice" : "audioDevice"
}
}