Http Server

First Steps

Introduction

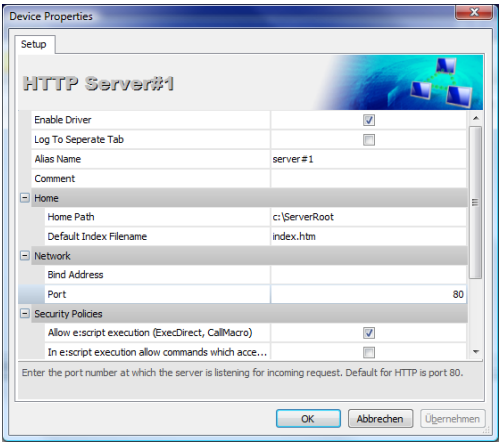
With v5.0, the e:cue Programmer introduces an embedded HTTP server. This enables the Programmer to deliver html content to any web browser and also to receive and process HTTP requests from clients. This powerful interface enables to setup of arbitrary remote user interfaces. Based on this technology numerous possibilities are given.

Basic Configuration

To set up a web server in the e:cue Programmer, perform the following steps:

1. Open the device manager window.
2. Select “Add Driver”.
3. In the upcoming window select “HTTP Server” (under “Servers”).
4. Press “Add”.

This will open a configuration dialog for the Server. Here you can configure the server to fit to your needs.



configuration dialog for the HTTP server

The following configuration options are available:

Base Options

Enable Driver – Only if this option is checked, the HTTP server will be active.

Log To Separate Tab – If you want the server to own a separate tab in the e:cue Programmer logbook, enable this option.

Alias Name – The driver's internal identification name. Can in most cases be left to standard.

Comment – This is an optional comment for the driver (e.g. for additional information).

Home

Home Path – The local directory where the HTTP server will search for content. This is the directory where you will have to put the website that you want to publish online.

Default Index Filename – The filename of the default index file. This will be shown if you browse to the web server's IP address.

Network

Bind Address – The IP Address to which the web server will be bound. Can be the computer's local network IP or the internet IP.

Port – By default, HTTP communicates via port 80. Only advanced users should adjust this value.

Security Policies

Allow e:script execution (ExecDirect, CallMacro) – This option must be checked to allow the HTTP server to allow any user interaction.

In e:script execution allow commands which … - By default, users cannot perform any commands that access the system, modify the current show or files. This is done for security reasons, as attackers may take over control of the system or destroy the show file via remote.

Both options do **not** override the Programmer security settings. If you disabled e:script system access inside the application options, this will still apply, regardless of what you set in the security policies.

These options should only be enabled in controlled areas as it may harm the security of your system! Use at your own risk!

Webcam Support

Webcam Device Driver – Selecting a webcam or capture device enables direct access to this device from HTML.

Remote Control Basics of the Programmer

Request Syntax

When the configuration is finished, the HTTP server is ready for access. As there is no webpage in the server home path, you cannot see anything. But you can already trigger actions in the Programmer using a web browser. This is done through special tags inside the URL. The basic syntax of an HTTP request is

```
http://ServerAddress/xmlget?RequestType=Request
```

or

```
http://ServerAddress/get?RequestType=Request
```

where *ServerAddress* is the IP Address or domain name of the HTTP server, *RequestType* is the type of request you want to send and *Request* is the request itself. Before we get to practical examples, the difference between the both lines above will be explained:

As you can see, the only difference between the lines is the use of `get` and `xmlget` respectively. This affects the style of the results that the programmer sends back to the browser after a request. Using `get`, the result is in plain text. With `xmlget`, the result will be send in XML, with the following layout:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<xml_ecue version="1.0">
    <request value="..." tag_id="..."/>
    <result value="1" error="0" message="OK"/>
    <result_list>
        <result_entry value="the first entry" />
  </result_list>
</xml_ecue>
```

XML-Tag	Description
<?xml ...	standard XML header
<xml_ecue>	protocol version number for e:cue HTML
<request>	the request like it arrived at the Programmer
<result>	'value' – an integer value for the result 'error' – nonzero value if an error occurred 'message' – error description in plain text
<result_list>	holds a list of <result_entry> – tags
<result_entry>	holds an attribute 'value' with results

Using get requires less result decoding effort but has a worse error handling. With xmlget, in contrast, decoding is not as simple but you can see more easily what went wrong in case an error occurred.

Empty spaces and various special chars are not allowed inside an HTTP request. You will have to transcode the request string to be conform with HTTP. The JavaScript command XmlHttpRequest automatically transcodes forbidden characters into the correct format automatically. You should use this command when you build up your own website for remote access.

Request Types

CallMacro

The `CallMacro` request will execute a macro in the programmer. The macro must exist inside the show.

```
http://ServerAddress/xmlget?CallMacro=Name
http://ServerAddress/get?CallMacro=Name
```

You can also pass up to 5 numeric parameters, separated with commas.

Examples:

```
http://ServerAddress/xmlget?CallMacro=MyMacro
http://ServerAddress/get?CallMacro=MyMacro,2,4,6
```

ExecCommand

The `ExecCommand` request will execute a single line e:cript command in the programmer.

```
http://ServerAddress/xmlget?ExecCommand=CommandString
http://ServerAddress/get?ExecCommand=CommandString
```

In CommandString you can add any valid e:script command (or command sequence) like you would use inside the Programmer. You can also define global variables.

Examples:

```
http://ServerAddress/xmlget?ExecCommand=StartCueList(QL(1));
http://ServerAddress/get?CallMacro=if (undefined(_x)) int _x; _x=10;
http://ServerAddress/xmlget?ExecCommand=printf("%d\n",_x);
```

AutoText

The AutoText request sends e:cue autotext to the Programmer. The Programmer sends back decoded autotext to the web browser as a result.

```
http://ServerAddress/xmlget?AutoText=AutoTextString
http://ServerAddress/get?AutoText=AutoTextString
```

Example:

```
http://ServerAddress/xmlget?AutoText=<cuelist 1 status>
```

Building a website for Remote Access

Using the HTTP requests by typing them directly into the browser's address bar is too complicated for real practical applications. It is of course possible to embed the requests into a website. This is what it's all about: You create your own website with your own control elements and put it online using the Programmer's integrated HTTP server. You can also refresh any status displays or control elements without having to reload the entire page. This is typically realized using the AJAX web development techniques.

The following tutorial will show how to build up a basic website for remote access. We will build a web page optimized for the Apple iPhone™. Using this page, it will be possible to control e:cue Programmer cuelists and also view some status information.

It will be assumed that you have basic knowledge about HTML and CSS (Cascading Style Sheets).

Tutorial

We begin with the main file, namely the index.html. First we create the basic layout for the website. We use a table with a fixed dimension so that it fits onto an iPhone display. It has various areas for buttons and info text.

index.html (interim)

```
<html>
<head>
    <title>e:cue Player</title>
</head>
<body>

<link rel="stylesheet" type="text/css" href="src/style.css">


<table width="320" height="365" class="container" border=0 cellpadding=0
cellspacing=0>
  <tr>
        <td colspan=3 class="header"><center>e:cue Player</center></td>
    </tr>
    <tr>
        <td colspan=3 class="label"><center>Q List</center></td>
    </tr>
    <tr class="info">
        <td><img src="images/btn_lt.png" ""></td>
        <td width="100%"><div id="qlistname"></div></td>
        <td><img src="images/btn_rt.png" onClick=""> </td>
    </tr>
    <tr class="label">
        <td colspan="3">
            <center>Current Q</center>
        </td>
    </tr>
    <tr class="info">
        <td colspan="3">
            <div class = "info" id="qname" style="height:25;"></div>
        </td>
    </tr>
    <tr class="label">
        <td colspan="3"> </td>
    </tr>
    <tr height="100%">
        <td colspan="3" style="vertical-align: bottom">
            <center>
                    <img src="images/btn_play.png" onClick="">
```

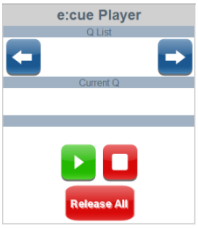
```
                        <img src="images/btn_stop.png" onClick="" ><br>
                        <img src="images/btn_rel_all.png" onClick="" >
                </center>
            </td>
        </tr>
    </table>
```

As you might have noticed, in index.html we included a file stlye.css with a few CSS style definitions. Without that file, our index.html would not look correct because it uses the defined styles for correct visual appearance. There are also some images for buttons.

src\style.css

```
    html, body {
        padding:0;
        margin:0;
    }

     .container{
         font-family: sans-serif;
         margin: 0px;
         padding: 0px;
         max-width: 400px;
         border: 2px solid #D0D0D0;
      }

     .label{
        background: #9fb1c2;
        text-align: center;
        color: #546471;
        text-shadow: #8fA1B2 2px 2px ;
        }

    .info{
        color: #333399;
        text-align: center;
        margin-bottom: 20px;
        }

    .play{
        text-align: center;
    }

    .header{
        margin:0;
        text-align: center;
        font-size: 150%;
        font-weight: bold;
        padding: 0;
        padding-bottom: 4px;
        background: #D0D0D0;
        color: #546471;
        text-shadow: #8fA1B2 2px 2px ;
    }
```

This will create a visually complete, but functionless website. It looks as follows:



There is already a stub in various elements for onclick events. For example, look at

```
<img src="images/btn_rel_all.png" onClick="" >
```

This is an image element with an onClick defined, but will not perform any action when it is clicked. We make now use of JavaScript to give life to the various elements. In the following JavaScript file we define many different functions for HTTP requests and Ajax updating techniques. Also, some base variables are defined that are used inside the script.

src\ecue.js

```
// Set this to false if you want to manually set you hostname & port
var use_default_host=true;

// Manually set the host name (only used if use_default_host is set to false)
// Make sure you set the correct port if you are configuring manually
var ip_address = "192.168.124.23";

//  Manually set the port # (Warning: Normally should be 80, but may need
//  to be different if another application, such as skype, interferes with
//  your HTTP server)
var port="8080";

//Decide what IP address to bind to
if (port== "") port = "80";
var baseURL = "http://" + ip_address + ":" + port + "/xmlget?";

if (use_default_host)
{
    baseURL = "http://" + location.host + "/xmlget?";
}

var current_cuelist = 1;


// Update the data on screen via AJAX(only if it has changed)
var lastlisthtml="";
var lastqhtml = "";
function updatestatus(){

  var listhtml;
  var atstring = "<cuelist " + current_cuelist + " name>";
  listhtml = getAutoText(atstring);
  if (listhtml == ""){
    current_cuelist--;
    updatestatus();
  }
  if (lastlisthtml != listhtml){
    document.getElementById("qlistname").innerHTML= listhtml;
    lastlisthtml = listhtml;
  }

  var qhtml;
  atstring = "<cuelist " + current_cuelist + " current>";
```

```
  qhtml = getAutoText(atstring);
  if (qhtml=="") { qhtml="--"};
  if (qhtml != lastqhtml){
    document.getElementById("qname").innerHTML=qhtml;
    lastqhtml = qhtml;
  }

  setTimeout("updatestatus()", 1000);
}

// Returns an autotext command from the server
function getAutoText(autotext){
  url = baseURL + "AutoText=" + autotext;
  var response = sendcommand(url);
  return (parseXML(response));
}

// Play specified cuelist
function pressplay(ql){
  url = baseURL+"ExecCommand=StartCuelist(QL(" + ql + "));"
  sendcommand(url);
  updatestatus();
}

// Select the next cuelist
function nextcuelist(){
    ++current_cuelist;

    updatestatus();
}

// Select the previous cuelist
function prevcuelist(){
    --current_cuelist;
    if (current_cuelist < 1) current_cuelist = 1;
    updatestatus();
}


// Stop the specified cuelist
function stop(ql){
  url = baseURL+"ExecCommand=StopCuelist(QL(" + ql + "));"
  sendcommand(url);
  updatestatus();
}

// Release all cuelists
function stopAll(){
  url = baseURL+"ExecCommand=ReleaseAll();"
  sendcommand(url);
  updatestatus();
}

function GetXmlHttpObject()
{
var xmlHttp=null;
try
  {
  // Firefox, Opera 8.0+, Safari
  xmlHttp=new XMLHttpRequest();
  }
catch (e)
  {
  // Internet Explorer
  try
    {
    xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
    }
  catch (e)
    {
```

```
    xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
  }
return xmlHttp;
}

// Sends a command to the e:cue 5.0 software
// returns XML data sent from the server
function sendcommand(url)
{
  xmlHttp=GetXmlHttpObject();
  if (xmlHttp==null)
    {
    alert ("Your browser does not support AJAX!");
    return;
    }
  //xmlHttp.onreadystatechange=stateChanged;
  xmlHttp.open("GET",url,false);
  xmlHttp.send(null);
  return xmlHttp.responseText;
}

function parseXML(text)
{
  var xmlDoc;

  try //Internet Explorer
    {
    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async="false";
    xmlDoc.loadXML(text);
    }
  catch(e)
    {
    try //Firefox, Mozilla, Opera, etc.
    {
    parser=new DOMParser();
    xmlDoc=parser.parseFromString(text,"text/xml");
    }
    catch(e)
    {
    alert(e.message);
    return;
    }
  }
  var result =
xmlDoc.getElementsByTagName("result_entry")[0].getAttribute('value');
  return result;
}

// Continually update visible info on screen
setTimeout("updatestatus()", 0);
```

The function `updatestatus` is called consecutively using the JavaScript command `setTimeout`. It uses Ajax to continuously update status information about the current cuelist and the name of the running cue.

The various HTTP server requests are encapsulated inside several functions like `pressplay(ql)`. Now all we have to do is to include the ecue.js into our index.html file. This is done by adding

```
<script src="src/ecue.js"></script>
```

into the index.html. Now we just make the fitting functions be called in our onClick events. Example:

```
<img src="images/btn_rel_all.png" onClick="stopAll();" >
```

If you now click on the image, the macro `stopAll`, which stops all cuelists, will be executed. This has to be done for every element. The final version of the index.html looks as follows:

index.html (final)

```
<html>
<head>
    <title>e:cue Player</title>
</head>
<body>

<script src="src/ecue.js"></script>

<link rel="stylesheet" type="text/css" href="src/style.css">


<table width="320" height="365" class="container" border=0 cellpadding=0
cellspacing=0/>
    <tr>
        <td colspan=3 class="header"><center>e:cue Player</center></td>
    </tr>

    <tr>
        <td colspan=3 class="label"><center>Q List</center></td>
    </tr>

    <tr class="info">
        <td><img src="images/btn_lt.png" onClick="prevcuelist();"></td>
        <td width="100%"><div id="qlistname"></div></td>
        <td><img src="images/btn_rt.png" onClick="nextcuelist();"> </td>
    </tr>

    <tr class="label">
        <td colspan="3">
            <center>Current Q</center>
        </td>
    </tr>

    <tr class="info">
        <td colspan="3">
            <div class = "info" id="qname" style="height:25;"></div>
        </td>
    </tr>

    <tr class="label">
        <td colspan="3"> </td>
    </tr>

    <tr height="100%">
        <td colspan="3" style="vertical-align: bottom">
            <center>
                    <img src="images/btn_play.png"
onClick="pressplay(current_cuelist);">
                    <img src="images/btn_stop.png"
onClick="stop(current_cuelist);" ><br>
                    <img src="images/btn_rel_all.png" onClick="stopAll();" >
            </center>
        </td>
    </tr>

</table>
```

Now finally the website is functional and can remotely control the e:cue Programmer. This is what it looks like:

With the blue buttons you can select a cuelist to control with the buttons below. You can start the selected cuelist with the green button and stop it with the red button. The big “Release All” button allows stopping all cuelists at the same time. In the middle you can see the name of the cue that is currently running.