CueServer™ Software Development Kit (SDK)

For CueServer Firmware Versions 5.1.0 and Later

Interactive Technologies

Interactive Technologies, Inc.
5295 Lake Pointe Circle Drive, Suite 100
Cumming, GA 30041 USA
Phone: 678-455-9019
Fax: 678-455-9071

Email: support@interactive-online.com
Web: http://interactive-online.com/

Printed in the United States of America.

Document D0440

Revised 6/14

Table of Contents

Introduction

The CueServer Software Development Kit (SDK) is designed for software programmers who want to use the powerful DMX playback capabilities of CueServer in their own applications.

CueServer has a complete set of open Application Programming Interfaces (APIs) that allow you to develop your own software that can interact with CueServer, request real-time information about the show it is playing back, send live data to CueServer that it should output to the connected devices, and load and store show data in CueServer's internal memory.

There are several mechanisms that software programmers can use to interact with CueServer:

- **HTTP CGI Interface** - CueServer includes several special URLs that are used to interact with virtually all aspects of its operation. Using these URLs you can issue real-time CueScript commands, send DMX data, request operating parameters and load and store show data of any kind.
- **Resources** - All CueServer show data is stored as a collection of "resources" in a database, either in CueServer's internal memory or in a CueServer .csd data file.
- **CueServer .csd File Format** - The .csd file format encapsulates an entire CueServer show, including cue data, event triggers and system configuration. You can read and/or generate these files and either save them to disk or upload and download them to a physical CueServer device.
- **XML Interface** - CueServer also includes an XML interface for interactive with CueServer from software objects that require XML formatted transactions (such as Adobe Flash).

This document describes the Public API features that are available in CueServer Firmware 5.1.0 and later. If you use a firmware version earlier than this, unexpected results may occur.

HTTP Interface

CueServer includes an embedded web server that responds to HTTP requests. In addition to the standard URLs that a user of the Web Interface would see, a special set of URLs are available in CueServer that can be used to fetch real-time information from CueServer, run CueScript commands, set operating parameters and to load and store CueServer resource data.

This chapter describes the various URLs that are available and their function.

Command throttling (IMPORTANT)

It is very imporant to employ a throttling mechanism when sending HTTP requests. New requests should only be sent after a response to the previous request has been received from CueServer. This mechanism allows CueServer to manage requests and responses gracefully and it prevents your code from sending a large number of commands blindly in a very short period of time (for example, moving a slider in a rapid fashion). Not implementing this mechanism could result in many dropped requests and commands, and theoretically to performace issues effecting other functions of CueServer due to extremely high CPU usage.

CueServer SDK URLs

/exe.cgi

The exe.cgi URL is used to execute CueScript commands.

When sending a exe.cgi request to a CueServer, you can use the following parameters:

- **cmd=<CueScript Command>**
  This parameter is used to specify the CueScript command to be executed by the CueServer. It is important to remember to URL encode the characters in the command (all spaces are must be converted to '+', and certain punctuation must be converted to the appropriate '%' notation).

- **usr=<User Number>** (optional)
  This optional parameter can be used to indicate which remote user is executing the command. By default, the user number is zero, which corresponds to the same user that may be using CueServer's web interface. User numbers 1, 2, 3 & 4 are also available to use in situations where multiple concurent users may be executing commands on the CueServer at the same time. Using different user numbers allows each user's context to remain separate.

- **def=<Default Playback>** (optional)
  This optional parameter can be used to specify the default playback number that should be used to execute the command. If not specified, the playback will be the last playback specified by the user (which is typically desired). In some cases, it is preferable to require that a command execute in a specific playback, which can be assigned using this parameter.

The response from the CueServer is the raw result of executing the command. Most commands return a single 2-byte integer, with `0x0000` meaning no error occurred.

For example, to send the command "Cue 1 Go", request the following URL:

```
http://<address>/exe.cgi?cmd=Cue+1+Go
```

To send the command "Channel 1>10 At FL" using User Number 1 and forcing CueServer to use Playback 3, request the following URL:

```
http://<address>/exe.cgi?cmd=Channel+1%3E10+At+FL&usr=1&def=3
```

/cmd.cgi

The cmd.cgi URL is used to execute CueScript commands with a redirect response from the CueServer that causes the web browser the originated the request to reload the original page that made the request.

This URL is designed to be used from a web page that executes a command and then requires the web page to reload itself.

When sending a cmd.cgi request to a CueServer, you use the following parameter:

- **cmd=<CueScript Command>**
  This parameter is used to specify the CueScript command to be executed by the CueServer. It is important to remember to URL encode the characters in the command (all spaces are must be converted to '+', and certain punctuation must be converted to the appropriate '%' notation).

The response from the CueServer is a HTTP Redirect (code 302) to the referrer page. This causes web browsers to reload the original page after the resonse is received.

The CueServer web interface uses this URL in several forms.

/pcmd.cgi

The pcmd.cgi URL is used to parse a CueScript command into it's English expansion.

When sending a cmd.cgi request to a CueServer, you use the following parameter:

- **cmd=<CueScript Command>**
  This parameter is used to specify the CueScript command to be parsed by the CueServer. It is important to remember to URL encode the characters in the command (all spaces are must be converted to '+', and certain punctuation must be converted to the appropriate '%' notation).

For example, to expand the command "T1C10AFL" to its English string, request the following URL:

```
http://<address>/pcmd.cgi?cmd=T1C10AFL
```

The plain text response send back to the requester will be

```
Time 1 Channel 10 At FL
```

/get.cgi

The get.cgi URL is used to fetch real-time information or resource data from a CueServer.

This URL can fetch many different objects, depending on the value of the `req` parameter in the URL. Some requests require additional parameters to be specified in the URL.

The following parameters are used with the get.cgi request. Only `req` is required, the other parameters are only used in certain requests.

- **req=<Request Type>**
  This is a two or three character string that specifies what is being requested from the CueServer. See the list below for details.
- **type=<Resource Type>** (optional)
  This is an optional number that corresponds to the Resource Type being requested by either a Get Resource or Get Resource List request.
- **id=<ID>** (optional)
  This is an optional number that corresponds to either a Resource ID or an object ID (such as with the Get Playback Info request).
- **page=<Page Number>** (optional)
  This is an optional number that is used to fetch multiple pages from the Get Resource List request.
- **usr=<User Number>** (optional)
  This is an optional number that is used with the Get Command Context request.
- **p=<ID>** (optional)
  This is an optional playback ID that is used with the Get Group Channel Values request.

The following specific request values are available for the `req` parameter:

BV (Button Values)

This request returns a block of 512 bytes that correspond to the current indicator values for the possible 512 buttons assignable to a CueServer.

CC (Command Context)

This request returns the following data structure that represents the requested command context. Use the `usr` parameter to specify which context to fetch (0..4).

```
typedef struct CommandContext {
   unsigned int     curPlayback;        // [  0] Current playback number (0..3)
   unsigned char    curTarget;          // [  2] Current command target
   unsigned char    isDMXTarget;        // [  3] Is the target a DMX collection
   unsigned int     fadeUpTime;         // [  4] Fade Up Time
   unsigned int     fadeDownTime;       // [  6] Fade Down Time
   unsigned char    reserved[8];        // [  8] Reserved
   unsigned char    selectMask[64];     // [ 16] Selected object mask
} CommandContext;
```

FF (Flicker Finder)

This request returns the following data structure that represents the result of analyzing the DMX Input stream for any changes since the last Flicker Finder request.

```
typedef struct FlickerFinder {
   unsigned char    flickerData[512];   // [  0] Change number array
   unsigned char    valueData[512];     // [512] Channel value array
} FlickerFinder;
```

If no DMX Input is present when the Flicker Finder data is requested, the response will be a single byte `0x00`.

IN (DMX Input)

This request returns the current DMX Input channel values being received by the CueServer's DMX Input port.

If CueServer is receiving 512 channels, then 512 bytes will be returned. If CueServer is receiving less than 512 bytes, then only that number of bytes is returned. If no DMX Input is being received at all, the response will be a single byte `0x00`.

LST (Get Resource List)

This request returns the list of Resource IDs from CueServer's database for a particular resource type. Use the `type` URL parameter to specify what Resource Type to fetch.

The response to this request is an array of the following 3-byte data structure:

```
typedef struct ResourceListItem {
   unsigned int     resID;                // Resource ID
   unsigned char    resSeed;              // Resource Seed
} ResourceListItem;
```

If no resources of the given type were found, the result is null.

If more than 400 resources of the given type were found, only the first 400 are returned. Use the `page` parameter in the URL to fetch the next page of 400 resources.

OUT (DMX Output)

This request returns the current DMX Output channel values being sent out the CueServer's DMX Output port.

The response to this request is always 512 bytes.

OV (Output Values)

This request returns a block of 512 bytes that correspond to the current digital output values for the possible 512 digital outputs assignable to a CueServer.

IMPORTANT: This is not the same as the DMX Output channel values.

PI (Playback Info)

This request returns the following data structure that describes the current detailed state of the requested playback fader. Use the `id` parameter to specify which playback fader to fetch (0..3).

```
typedef struct PlaybackInfo {
   unsigned char    playback;             // Playback number (1..4)
   unsigned char    runMode;              // Run Mode (0 = Normal, 1 = Stopped)
   unsigned char    outputLevel;          // Output level (0..255)
   unsigned char    combineMode;          // Combine Mode
   unsigned int     fadeTimer;            // Remaining fade time in progress
   unsigned int     followTimer;          // Remaining follow time in progress
   unsigned long    streamTimer;          // Stream playback position
   unsigned int     currentCue;           // Cue currently playing
   unsigned int     nextCue;              // Next cue (0 = None, 1 = Cue 0.1)
   unsigned long    fadeTimes;            // Fade Times (up/down) for next cue
   unsigned int     followTime;           // Follow Time for next cue
   unsigned int     linkCue;              // Linked cue for next cue
   unsigned char    reserved[8];          // Reserved
   unsigned char    currentName[32];      // Name of current cue
   unsigned char    nextName[32];         // Name of next cue
} PlaybackInfo;
```

PS (Playback Status)

This request returns the following data structure that describes the current overall state of all playback faders:

```
typedef struct PlaybackInfo {
   unsigned int     current1;              // Playback 1 Current Cue
   unsigned int     next1;                 // Playback 1 Next Cue
   unsigned int     reserved1[4];          // Reserved
   unsigned int     current2;              // Playback 2 Current Cue
   unsigned int     next2;                 // Playback 2 Next Cue
   unsigned int     reserved2[4];          // Reserved
   unsigned int     current3;              // Playback 3 Current Cue
   unsigned int     next3;                 // Playback 3 Next Cue
   unsigned int     reserved3[4];          // Reserved
   unsigned int     current4;              // Playback 4 Current Cue
   unsigned int     next4;                 // Playback 4 Next Cue
   unsigned int     reserved4[4];          // Reserved
} PlaybackInfo;
```

PO (Playback Output)

This request returns the current output values and DMX source data for each DMX channel in the system. The following data structure is returned:

```
typedef struct PlaybackOutput {
   unsigned char    channelValues[512];    // Channel Values
   unsigned char    dmxSource[512];        // DMX Sources
} PlaybackOutput;
```

P1 .. P4 (Playback Channel Values)

This request returns the current output values and DMX source data for each DMX channel in a specified playback fader. The following data structure is returned:

```
typedef struct PlaybackOutput {
   unsigned char    channelValues[512];    // Channel Values
   unsigned char    dmxSource[512];        // DMX Sources
} PlaybackOutput;
```

RES (Get Resource)

This request returns a resource from CueServer's database.

Use this command with the `type` and `id` URL parameters to specify which resource data to fetch.

The returned result will be a block of 512 bytes for the specified resource.

If the resource could not be found, a single byte corresponding to an error code is returned.

SI (System Info)

This request returns the following data structure that describes the current overall state of the CueServer:

```
typedef struct SystemInfo {
    unsigned char    serial[16];            // Serial Number
    unsigned char    deviceName[24];        // Device Name
    unsigned char    firmwareVersion[12];   // Firmware Version
    unsigned char    timeStr[24];           // Time String
    unsigned char    model;                 // 0 = Unknown, 1 = CS-800, 2 = CS-810, etc.
    unsigned char    hasPassword;           // 0 = No, 1 = Yes
} SystemInfo;
```

GRP (Group Channel Values)

This request returns the value of a group of channels. Use it to detect DMX values for all channels in a group, or whether the values are the same for all channels (useful for group button indicator or group slider).

Use this command with the `id` and `p` URL parameters to specify group number and desired playback fader. Use `p=0` for Output values. For example, to detect values for group 1 on playback fader 1, send the following request:

`http://<address>/get.cgi?req=grp&id=1&p=1`

The returned result will be 2 byte signed integer. If all channels in the group have no DMX value (not zero), or if one or more channels is different from the rest of the group, the returned value will be -1 (0xFFFF). If all channels have the same valid value (including zero), then the response will be the DMX value (f.e. 0x0A00 for decimal DMX value 10).

If there is only one channel in the group, the returned result will be a DMX value, unless the channel has no value, which will return -1.

/set.cgi

The set.cgi URL is used to store real-time information or resource data to a CueServer.

This URL can set many different objects, depending on the value of the `dst` parameter in the URL. Some set requests require additional parameters to be specified in the URL.

The following parameters are used with the set.cgi request. Only `dst` is required, the other parameters are only used in certain requests.

- **dst=<Destination>**
  This is a two or three character string that specifies what data is being set in the CueServer. See the list below for details.
- **type=<Resource Type>** (optional)
  This is an optional number that corresponds to the Resource Type being set by a Store Resource request.
- **id=<ID>** (optional)
  This is an optional number that corresponds to a Resource ID.
- **del=<ID>** (optional)
  This is an optional number that corresponds to a Resource ID that is being deleted.

The following specific request values are available for the `dst` parameter:

P1 .. P4 (Playback Channel Values)

This request will set the current values of one of the playback faders to the given buffer of 512 bytes.

The result of setting a playback's channel values in this manner is immediate, with no fade time used.

If more or less than 512 bytes are provided, no channels will be set and this will return an error code.

If the request was successful, a single byte `0x00` will be returned.

RES (Set Resource)

This request will set the data of the specified resource to the given bytes.

Use the `type` and `id` URL parameters to specify the Resource Type and ID of the resource data to store.

If the optional `del` URL parameter is specified, then the resource with this ID is deleted immediately before the new resource data is stored. Using the `del` parameter with the `id` parameter is useful when renumbering resources.

/xml.cgi

CueServer provides a simple XML interface for requesting and setting a subset of objects. This mechanism was developed for simplyfying the integration with Adobe Flash projects.

The details of this URL are available from Interactive Technologies technical support if needed.

Resources

About Resources

All show data is stored in CueServer as a series of “resources”. A resource is a data object that is identified by a Type and ID (and in the case of CueStream resources, also a Time). The definition of every object in CueServer, such as a Cue, Timer, Macro, Button, etc., is stored in CueServer’s memory as individual resources.

Resource Types generally correspond to the object that the resource represents. For example, each Timer defined in a show is stored in a Timer resource.

The Resource ID is a unique number that identifies multiple resources of the same type. For example, the data for Macro 7 is stored in a resource of type Macro, with an ID of 7.

You can think of resources as CueServer’s file system. There are no directories, just a database of resources organized by Type and ID. When a new show is loaded into CueServer, the database is replaced with new resources, and CueServer begins playing a new show, and responding to different buttons, executing new timers, etc.

Every resource contains exactly 512 bytes of data, however, not all resource types use all 512 bytes. Each resource type is defined by a data structure that specifies where each data field is located in the 512 byte data block.

The CueServer .csd file format is largely a list of CueServer data resources. Also, the CGI API for CueServer allows other software to load and store resources of any type on the fly while a CueServer is running.

Resource Definitions

This section provides details about each resource type.

CueInfo Resource

The CueInfo resource provides details about a cue, including it’s name, fade and follow times, link, channel mask, CueScript action and more. It does not, however include the channel values. Channel values are stored in a separate CueData resource with the same ID.

The Resource ID of a CueInfo resource is ten times (x10) the Cue Number. For example, Cue 1 is stored in ID 10. Cue 2.5 uses resource 25. Valid cue numbers range from 0.1 through 6499.9, which corresponds to IDs from 1 through 64999.

The following C data structure defines the fields in the CueInfo resource:

```
#define RESTYPE_CUEINFO                0x10   // Resource type for CueInfo resources
#define CUE_LIST_MAX_SIZE              2000   // Maximum number of cues in the cue list
#define MAX_CUE_NUMBER                 64999  // Maximum cue number = 6499.9

typedef struct CueInfo {
   unsigned int     cueNumber;          // [  0] Cue number (1..64999)
   unsigned int     reserved1;          // [  2] Must be 0xFFFF
   unsigned int     followTime;         // [  4] Follow time
   unsigned int     linkCue;            // [  6] Follow cue
   unsigned char    type;               // [  8] Type: 0 = Normal, 1 = Stream
   unsigned char    flags;              // [  9] Flags: Bit 0 = Decimal Adjusted Cue Number
   unsigned char    followMode;         // [ 10] Follow Mode for Streaming Cues
   unsigned char    reserved2[11];      // [ 11] Reserved
   unsigned long    fadeTimes;          // [ 22] Split fade times (MSW/LSW = Up/Down)
   unsigned long    streamDuration;     // [ 26] Total time of stream (ms)
   unsigned int     streamBlocks;       // [ 30] Number of stream blocks in cue
   unsigned char    name[32];           // [ 32] Name
   unsigned char    mask[64];           // [ 64] Bitmask
   unsigned char    command[128];       // [128] Command string
} CueInfo;
```

CueData Resource

The CueData resource is used in conjunction with “normal” cues. For each CueInfo resource with `type = 0`, a corresponding CueData resource with the same ID will also be in the database that includes the channel levels for the cue.

The following C data structure defines the CueData resource:

```
#define RESTYPE_CUEDATA                0x11   // Resource type for CueData resources

typedef struct CueData {
   unsigned char    levels[512];        // [  0] Channel levels
} CueData;
```

CueStream Resource

The CueStream resource is used in conjunction with “Streaming” cues. For each CueInfo resource with `type = 1`, one or more CueStream resources with the same ID will also be in the database that represent various snapshots throughout the recorded stream. CueStream resources are the only resources that have multiple records with the same resource Type and ID (but with unique Times).

So, for example, if Cue 1 is a Streaming Cue, then a single CueInfo resource with ID 10 will appear in the database and one or more CueStream resources with ID 10 will also appear in the database. Each CueStream resource will be recorded with a different Time property. Resource Times are expressed in milliseconds.

The following C data structure defines the CueStream resource:

```
#define RESTYPE_CUESTREAM               0x07   // Resource type for CueStream resources

typedef struct CueStream {
   unsigned char    levels[512];        // [  0] Channel levels
} CueStream;
```

Group Resource

The Group resource provides details about a Group, including it’s name and a bitmask indicating which channels are part of the group.

The Resource ID of a Group resource is equal to the group number. Valid group numbers range from 1 through 999.

The following C data structure defines the fields in the Group resource:

```
#define RESTYPE_GROUP                   0x12   // Resource type for Group resources
#define GROUP_LIST_MAX_SIZE             100    // Maximum number of groups in the group list
#define MAX_GROUP_NUMBER                999    // Maximum group number

typedef struct GroupResource {
   unsigned int     groupNumber;        // [  0] Group number
   unsigned char    reserved1[14];      // [  2] Reserved
   unsigned char    mask[64];           // [ 16] Bitmask
   unsigned char    name[32];           // [ 80] Name
} GroupResource;
```

Button & Contact Resources

The Button or Contact resource provides details about a Button or Contact, including it's name, its trigger parameters and the CueScript actions it should perform.

The Resource ID of a Button or Contact resource is equal to the button or contact number. Valid resource IDs range from 1 through 512. The only difference between a button or contact resource is its Resource Type.

The following C data structure defines the fields in the Button and/or Contact resource:

```
#define RESTYPE_BUTTON                  0x1B   // Resource type for Button resources
#define MAX_BUTTON_NUMBER               512    // Maximum button number
#define BUTTON_LIST_MAX_SIZE            512    // Max number of groups in the button list
#define RESTYPE_CONTACT                 0x1C   // Resource type for Contact resources
#define CONTACT_LIST_MAX_SIZE           512    // Max number of groups in the contact list
#define MAX_CONTACT_NUMBER              512    // Maximum contact number

typedef struct ButtonResource {
   unsigned int     id;                // [  0] Button ID (1..512)
   unsigned char    type;              // [  2] 0 = button, 1 = contact
   unsigned char    function;          // [  3] 0 = momentary, 1 = state-driven, etc.
   unsigned int     param;             // [  4] Channel/Group/Preset Number
   unsigned long    fadeTimes;         // [  6] Split Fade Times
   unsigned char    level;             // [ 10] Level
   unsigned char    flags;             // [ 11] Flags
   unsigned int     group;             // [ 12] Group
   unsigned char    reserved[10];      // [ 14] Reserved
   unsigned char    indicatorMode;     // [ 24] 0 = none
   unsigned char    indicatorFlags;    // [ 25] Bit 0 = Use Off Color, Bit 1 = Use On Color
   unsigned int     indicatorParam;    // [ 26] Channel/Group/Preset Number
   unsigned int     indicatorGroup;    // [ 28] Group
   unsigned char    onColor;           // [ 30] On Color
   unsigned char    offColor;          // [ 31] Off Color
   unsigned char    name[32];          // [ 32] Name
   unsigned char    actionA[128];      // [ 64] Action string A (press, close, etc.)
   unsigned char    actionB[128];      // [192] Action string B (release, open, etc.)
} ButtonResource;
```

Timer Event Resource

The Timer resource provides details about a Timer Event, including it's name, the details about when the timer should fire and the CueScript action that should be taken.

The Resource IDs of Timer resources are always sequential and start with ID 0.

The following C data structure defines the fields in the Timer resource:

```
#define RESTYPE_TIMER                   0x13   // Resource type for Timer resources
#define TIMER_LIST_MAX_SIZE             100    // Maximum number of timers in the timer list

typedef struct TimerData {
   intTimerResourceastro;               // [  0] 0 = regular, 1 = sunrise, 2 = sunset
   int              time;               // [  2] Time of the day (in minutes since midnight)
   int              dotw;               // [  4] Days of the week mask [SMTWTFS-]
   int              offset;             // [  6] Offset from astro time (+/-), in minutes
   int              startDay;           // [  8] First day number to activate (0 = Any)
   int              endDay;             // [ 10] Last day number to activate (0 = Same)
   int              reserved[2];        // [ 12] Reserved
   unsigned char    name[32];           // [ 16] Name
   unsigned char    cmd[128];           // [ 48] Command string
} TimerResource;
```

Timecode Event Resource

The Timecode resource provides details about a Timecode Event, including it's name, the SMPTE trigger time and the CueScript action that should be taken.

The Resource IDs of Timecode resources are always sequential and start with ID 0.

The following C data structure defines the fields in the Timecode resource:

```
#define RESTYPE_TIMECODE                0x15   // Resource type for Timecode resources
#define TIMECODE_LIST_MAX_SIZE          500    // Max number of items in the timecode list

typedef struct TimecodeResource {
   unsigned long    time;               // [  0] Timecode of event
   unsigned char    reserved[12];       // [  4] Reserved
   unsigned char    name[32];           // [ 16] Name
   unsigned char    cmd[128];           // [ 48] Command string
} TimecodeResource;
```

Macro Resource

The Macro resource provides details about a Macro, including it's name and the CueScript action that should be performed.

The Resource ID of a Macro resource is equal to the Macro number. Valid macro numbers range from 1 through 65439.

The following C data structure defines the fields in the Macro resource:

```
#define RESTYPE_MACRO                   0x16   // Resource type for Macro resources
#define MACRO_LIST_MAX_SIZE             200    // Maximum number of macros in the macro list
#define MAX_MACRO_NUMBER                65439  // Maximum group number

typedef struct MacroResource {
   unsigned int     id;                 // [  0] ID Number of Macro
   unsigned char    showInMenu;         // [  2] Show in menu
   unsigned char    reserved[13];       // [  3] Reserved
   unsigned char    name[32];           // [ 16] Name
   unsigned char    cmd[128];           // [ 48] Command string
} MacroResource;
```

DMX Input Trigger Resource

The DMX Input Trigger resource provides details about a DMX Input Event, including it's name, trigger parameters and the CueScript actions that should be taken.

The Resource IDs of DMX Input Trigger resources are always sequential and start with ID 0.

The following C data structure defines the fields in the DMX Input Trigger resource:

```
#define RESTYPE_DMXTRIGGER              0x17   // Resource type for DMX Trigger resources
#define DMXTRIGGER_LIST_MAX_SIZE        256    // Max number of items in the DMX Trigger list

typedef struct DMXTriggerResource {
   unsigned char    rangeLow;           // [  0] Low end of range
   unsigned char    rangeHigh;          // [  1] High end of range
   unsigned int     channel;            // [  2] DMX Channel (0 - 511)
   unsigned char    reserved[12];       // [  4] Reserved
   unsigned char    name[32];           // [ 16] Name
   unsigned char    enterCmd[128];      // [ 48] Command string for enter event
   unsigned char    exitCmd[128];       // [176] Command string for exit event
} DMXTriggerResource;
```

CueServer .csd File Format

CueServer uses the .csd file format to load and save show file databases into and out of CueServer's internal memory.

You can use this file format to either parse an existing show file or create one yourself.

CueServer can read and/or write these files directly from it's web interface, or through the CueServer Admin tool (an application program available for both Mac and Windows) or programmatically via an HTTP mechanism.

File Specifications

The CSD file format is designed to store an array of "CueServer resources". A resource is a 512-byte block of data that has an associated Type and ID (and optionally a timestamp, that is used only for resources of type "stream"). See the Resources chapter in this manual for details.

Resource Types may be CueInfo, CueData, Macro, Timer, Button, etc. The resource ID usually corresponds to the resource type, for example, Macro 3 has a resource ID of 3, Cue 1.0 has a resource ID of 10.

The file is read and written by "lines". There are two types of lines supported.

Comment Line

A comment line starts with a '`#`' character and ends with a line-feed (`0x0A`) byte. The data between the `#` and the `LF` is ignored and may be of any length.

All CSD files start with a comment line of the following format:

```
# CueServer Data File (xxxx)
```

The `xxxx` above is replaced with the firmware version of the CueServer that produced the file.

Resource Data Line

A resource data line stores the 512-byte resource data as well as the resource's Type, ID and optionally it's Timestamp.

Each data line is formatted as follows:

<Resource Type><Resource ID><Resource Time>'`$`'<Resource Data>'`!`'

- **Resource Type**
  This is an ASCII printed byte (two ASCII characters) that specify the Resource Type for the line. Valid values are from "00" through "1F".

- **Resource ID**
  This is an ASCII printed 2-byte word (four ASCII characters) that specify the Resource ID for the line. The MSB is printed first. Values from “0000” through “FFFF” are valid.
- **Resource Time**
  This is an ASCII printed 4-byte long (eight ASCII characters) that specify the Resource Time for the line. The MSB is printed first. This field may contain values “00000000” through “FFFFFFFF”.
- **$**
  This is a single byte 0x24.
- **Resource Data**
  This is the 512 bytes in the resource. They are not ASCII encoded.
- **!**
  This is a single byte 0x21.

Given the fields above, every data line in the file is exactly 528 bytes long.

The file ends at the end of the last resource line or comment line.

When reading this file in, if the first comment is improperly formatted or if any of the markers in the file are incorrect (the ‘$’ and the ‘!’), the file is considered bad.