

## VJLIVE3 SDK

VJLive3 provide webservice interface, webservice description address:

<http://nagasoft.cn/download/ws/vjlive3/VJLiveService.wsdl>.

### Document Conventions

1. String use UTF-8 encoding.
2. Webservice access password support plain text or md5 text (32 lowercase characters).

### Structure List

```
struct ns__playlist_item{
    char* name;
    char* url;           //supported format:
                        //File, eg: c:\some.wmv,/root/some.wmv
                        //mms, eg:mms://nagasoft.cn/c2
                        //mms pull, eg:mms://localhost:8080
                        // http ts, eg:https://localhost:1234
                        //vje pull, eg:vje://localhost:1234
    int duration;        //seconds
    int start_pos;       //seconds
    int end_pos;         //seconds
    int buffer_time;     //seconds
    int order_type;      // set to 0 for now.
};

struct playlist{
    struct ns__playlist_item* __ptr;
    int __size;
};

struct cur_play_info{
    char* name;
    int time;            //current play position, seconds
    int duration;        //seconds
};

struct ns__source_channel{
    char* name;          //channel name, unique.
    int cid;             //channel id, unique.
    int mdc;             //Max directly connections.
```

```

// formula: MDC = Server bandwidth * 0.9 / bitrate.
int buffer;           //buffer size, seconds.
int state;            //0=stopped,1=running.
struct playlist pl;   //
int curplay;          //>=0
struct cur_play_info cpi; //
bool allow_mirror;    //
bool push_first;      //
};

struct source_channel_list{
    struct ns__source_channel* __ptr;
    int __size;
};

struct ns__mirror_channel{
    char* name;        // channel name, unique.
    int cid;           // channel id, unique.
    int mdc;           // Max directly connections.
                        // formula: MDC = Server bandwidth * 0.9 / bitrate.
    int buffer;        // buffer size, seconds.
    int state;         //0=stopped,1=running.
};

struct mirror_channel_list{
    struct ns__mirror_channel* __ptr;
    int __size;
};
```

### Return values

WS_SUCCESS	0
WS_ERROR_INVALID_PWD	-1
WS_ERROR_UNKOWN	-2
WS_ERROR_INVALID_PARAM	-3
WS_ERROR_CREATE_DIR_FAILED	-4
WS_ERROR_PATH_NOT_EXIST	-5
WS_ERROR_TASK_NOT_EXIT	-6
WS_ERROR_CHANNEL_NAME_EXIST	-7
WS_ERROR_INVALID_CHANNEL_LIC	-8
WS_ERROR_INVALID_TRACKER	-9
WS_ERROR_INVALID_CHANNEL_ID	-10
WS_ERROR_CHANNEL_ID_EXIST	-11
WS_ERROR_CHANNEL_NOT_EXIST	-12
WS_ERROR_REQUEST_TRIAL_CHANNEL_FAILED	-13

---

WS_ERROR_WAIT_LOGIN_TO_NATSRV	-14
WS_ERROR_TRACKER_REPORT_FAILED	-15
WS_ERROR_NO_RUNTIME_LOG	-16
WS_ERROR_OUT_OF_MEMORY	-17

## Function List

### 1. getChannelsList

```
getChannelsList(  
    char* pwd, //access password  
    struct getChannelsListResponse& out //output, see output parameter for detail  
);
```

Output parameter:

```
struct getChannelsListResponse{  
    struct source_channel_list source; //source channel list  
    struct mirror_channel_list mirror; //mirror channel list  
    int result; //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS  
WS\_ERROR\_INVALID\_PWD  
WS\_ERROR\_UNKOWN

### 2. getSourceChannel

```
getSourceChannel(  
    char* pwd, //access password  
    char* name, //source channel name  
    struct getSourceChannelResponse& out // output, see output parameter for detail  
);
```

Output parameter:

```
struct getSourceChannelResponse{  
    struct ns__source_channel sc; //source channel info  
    int result; // return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS  
WS\_ERROR\_INVALID\_PWD  
WS\_ERROR\_UNKOWN  
WS\_ERROR\_INVALID\_PARAM  
WS\_ERROR\_CHANNEL\_NOT\_EXIST

## 3. getMirrorChannel

```
getMirrorChannel(  
char* pwd, //access password  
char* name, //mirror channel name  
struct getMirrorChannelResponse& out //output, see output parameter for detail  
);
```

Output parameter:

```
struct getMirrorChannelResponse{  
    struct ns__mirror_channel mc; //mirror channel info  
    int result; //return value, see return values for detail  
};
```

Return values:

```
WS_SUCCESS  
WS_ERROR_INVALID_PWD  
WS_ERROR_UNKOWN  
WS_ERROR_INVALID_PARAM  
WS_ERROR_CHANNEL_NOT_EXIST
```

## 4. createSourceChannel

```
createSourceChannel(  
char* pwd, //access password  
struct create_source_channel_params* in, //input, see input parameter for detail  
struct createSourceChannelResponse& out //output, see output parameter for  
detail  
);
```

Input parameter:

```
struct create_source_channel_params{  
    char* name; //channel name, unique.  
    int mdc; //max directly connections.  
    int buffer; //buffer size, seconds.  
    bool allow_mirror; //  
    bool push_first; //  
    struct xsd__base64Binary lic_data; //channel license data, from vvc file.  
    bool request_trial_channel; //request channel channel  
};
```

Output parameter:

```
struct createSourceChannelResponse{  
    struct ns__source_channel sc; //source channel info  
    int result; //return value, see return values for  
detail
```

```
};
```

Return values:

```
WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM
WS_ERROR_CHANNEL_NAME_EXIST
WS_ERROR_REQUEST_TRIAL_CHANNEL_FAILED
WS_ERROR_INVALID_CHANNEL_LIC
WS_ERROR_INVALID_TRACKER
WS_ERROR_INVALID_CHANNEL_ID
WS_ERROR_CHANNEL_ID_EXIST
```

Remarks:

lic\_data must valid if request\_trial\_channel is false.

#### 5. createMirrorChannel

```
createMirrorChannel(
char* pwd, //access password
struct create_mirror_channel_params* in, //input, see input parameter for detail
struct createMirrorChannelResponse& out //output, see output parameter for detail
);
```

Input parameter:

```
struct create_mirror_channel_params{
    char* name; //channel name, unique.
    int cid; //channel id, unique.
    int mdc; //max directly connections.
    int buffer; //buffer size, seconds.
};
```

Output parameter:

```
struct createMirrorChannelResponse{
    struct ns__mirror_channel mc; //mirror channel info
    int result; //return value, see return values for detail
};
```

Return values:

```
WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM
WS_ERROR_CHANNEL_NAME_EXIST
```

WS\_ERROR\_INVALID\_CHANNEL\_ID

WS\_ERROR\_CHANNEL\_ID\_EXIST

#### 6. startChannel

startChannel (

```
char* pwd, //access password
struct startChannelResponse& out //output, see output parameter for detail
);
```

Output parameter:

```
struct startChannelResponse{
    int result; //return value, see return values for detail
};
```

Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN

WS\_ERROR\_INVALID\_PARAM

WS\_ERROR\_CHANNEL\_NOT\_EXIST

WS\_ERROR\_WAIT\_LOGIN\_TO\_NATSRV

#### 7. stopChannel

stopChannel (

```
char* pwd, //access password
struct stopChannelResponse& out //output, see output parameter for detail
);
```

Output parameter:

```
struct stopChannelResponse{
    int result; //return value, see return values for detail
};
```

Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN

WS\_ERROR\_INVALID\_PARAM

WS\_ERROR\_CHANNEL\_NOT\_EXIST

#### 8. deleteChannel

deleteChannel (

```
char* pwd, //access password
struct deleteChannelResponse& out //output, see output parameter for detail
);
```

```
);
```

Output parameter:

```
struct deleteChannelResponse{  
    int result; //return value, see return values for detail  
};
```

Return values:

```
WS_SUCCESS  
WS_ERROR_INVALID_PWD  
WS_ERROR_UNKOWN  
WS_ERROR_INVALID_PARAM  
WS_ERROR_CHANNEL_NOT_EXIST
```

#### 9. renameChannel

```
renameChannel(  
    char* pwd, //access password  
    char* name, //old channel name  
    char* newname, //new channel name  
    struct renameChannelResponse& out //output, see output parameter for detail  
);
```

Output parameter:

```
struct renameChannelResponse{  
    int result; //return value, see return values for detail  
};
```

Return values:

```
WS_SUCCESS  
WS_ERROR_INVALID_PWD  
WS_ERROR_UNKOWN  
WS_ERROR_INVALID_PARAM  
WS_ERROR_CHANNEL_NOT_EXIST  
WS_ERROR_CHANNEL_NAME_EXIST
```

#### 10. setSourceChannelPlaylist

```
setSourceChannelPlaylist(  
    char* pwd, //access password  
    char* name, //channel name  
    struct playlist* pl, //  
    struct setSourceChannelPlaylistResponse& out //output, see output parameter for  
    detail  
);
```

Output parameter:

```
struct setSourceChannelPlaylistResponse{  
    int result;                //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS  
WS\_ERROR\_INVALID\_PWD  
WS\_ERROR\_UNKOWN  
WS\_ERROR\_INVALID\_PARAM  
WS\_ERROR\_CHANNEL\_NOT\_EXIST

11. setSourceChannelCurPlay

```
setSourceChannelCurPlay(  
    char* pwd,                //access password  
    char* name,               //channel name  
    int curplay,              //current play, start from 0.  
    struct setSourceChannelCurPlayResponse& out //output, see output parameter for  
    detail  
);
```

Output parameter:

```
struct setSourceChannelCurPlayResponse{  
    int result;                //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS  
WS\_ERROR\_INVALID\_PWD  
WS\_ERROR\_UNKOWN  
WS\_ERROR\_INVALID\_PARAM  
WS\_ERROR\_CHANNEL\_NOT\_EXIST

12. getSourceChannelPlaylist

```
getSourceChannelPlaylist(  
    char* pwd,                //access password  
    char* name,               //channel name  
    struct getSourceChannelPlaylistResponse& out //output, see output parameter for  
    detail  
);
```

Output parameter:

```
struct getSourceChannelPlaylistResponse{  
    struct playlist pl;        //
```



```
int curplay;           //current play, start from 0.
int result;            //return value, see return values for detail
};
```

Return values:

```
WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM
WS_ERROR_CHANNEL_NOT_EXIST
```

### 13. getSourceChannelCurPlayInfo

```
getSourceChannelCurPlayInfo(
char* pwd,                //access password
char* name,              //channel name
struct getSourceChannelCurPlayInfoResponse& out //output, see output parameter for detail
);
```

Output parameter:

```
struct getSourceChannelCurPlayInfoResponse{
    struct cur_play_info cpi;    //current play info.
    int result;                 //return value, see return values for detail
};
```

Return values:

```
WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM
WS_ERROR_CHANNEL_NOT_EXIST
```

### 14. getChannelStatus

```
getChannelStatus(
char* pwd,                //access password
char* name,              //channel name
struct getChannelStatusResponse& out //output, see output parameter for detail
);
```

Output parameter:

```
struct channel_status{
    int cid;                //channel id
    int online;             //
    int cdc;               //the player counts which directly connect to server.
```

```
char* start_dtime;      //start time, format: 2010-09-09 08:00:00
int howlong_started;    //seconds.
double upload_speed;    //KB/S
double total_upload;    // MB
double download_speed;  // KB/S
double total_download;  // MB
int cpu_usage;          // 0~100
int memory_usage;       // MB
};

struct getChannelStatusResponse{
    struct channel_status cs;    //
    int result;                  //return value, see return values for detail
};
```

Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN

WS\_ERROR\_INVALID\_PARAM

WS\_ERROR\_CHANNEL\_NOT\_EXIST

#### 15. setSourceChannelAccessCheck

```
setSourceChannelAccessCheck(
    char* pwd,                //access password
    char* name,               //channel name
    struct access_check* in,   //input, see input parameter for detail
    struct setSourceChannelAccessCheckResponse& out //output, see output
    parameter for detail
);
```

Input parameter:

```
struct access_check{
    int type;                //Access check type.
    char* content;           //The content of access check, set to empty to clear the setting.
};
```

type=1, simple password access check, content=md5 of password (Upper 32 characters)

type=2, user+password access check, content=the url of check web page.

type=3, domain access check, content=the url of domain check web page.

Please refer to Appendix 1 for the requests and responses of the "user+password" or domain check web page.

Output parameter:

```
struct setSourceChannelAccessCheckResponse{  
    int result;           //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS  
WS\_ERROR\_INVALID\_PWD  
WS\_ERROR\_UNKOWN  
WS\_ERROR\_INVALID\_PARAM  
WS\_ERROR\_CHANNEL\_NOT\_EXIST  
WS\_ERROR\_TRACKER\_REPORT\_FAILED

16.      getSourceChannelAccessCheck

```
getSourceChannelAccessCheck(  
    char* pwd,           //access password  
    char* name,          //channel name  
    struct getSourceChannelAccessCheckResponse& out //output, see output  
    parameter for detail  
);
```

Output parameter:

```
struct getSourceChannelAccessCheckResponse{  
    struct access_check ac;    //  
    int result;               //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS  
WS\_ERROR\_INVALID\_PWD  
WS\_ERROR\_UNKOWN  
WS\_ERROR\_INVALID\_PARAM  
WS\_ERROR\_CHANNEL\_NOT\_EXIST

17.      setSourceChannelAD

```
setSourceChannelAD(  
    char* pwd,           //access password  
    char* name,          //channel name  
    char* ad,           //read from vva file. Use "VJADMaker" to create vva files.  
    struct setSourceChannelADResponse& out //output, see output parameter for  
    detail  
);
```

Output parameter:

```
struct setSourceChannelADResponse{  
    int result;          //return value, see return values for detail  
};
```

Return values:

```
WS_SUCCESS  
WS_ERROR_INVALID_PWD  
WS_ERROR_UNKOWN  
WS_ERROR_INVALID_PARAM  
WS_ERROR_CHANNEL_NOT_EXIST  
WS_ERROR_TRACKER_REPORT_FAILED
```

#### 18. getSourceChannelAD

```
getSourceChannelAD(  
    char* pwd,           //access password  
    char* name,          //channel name  
    struct getSourceChannelADResponse& out //output, see output parameter for  
    detail  
);
```

Output parameter:

```
struct getSourceChannelADResponse{  
    char* ad;            //The content of AD.  
    int result;          //return value, see return values for detail  
};
```

Return values:

```
WS_SUCCESS  
WS_ERROR_INVALID_PWD  
WS_ERROR_UNKOWN  
WS_ERROR_INVALID_PARAM  
WS_ERROR_CHANNEL_NOT_EXIST
```

#### 19. setSourceChannelProperty

```
setSourceChannelProperty(  
    char* pwd,           //access password  
    char* name,          //channel name  
    struct source_channel_property* in, //input, see input parameter for detail  
    struct setSourceChannelPropertyResponse& out //output, see output parameter for  
    detail  
);
```

Input parameter:

```
struct source_channel_property{
```

```
int mdc;           //max directly connections.  
int buffer;        //buffer size, in seconds.  
bool allow_mirror; //  
bool push_first;   //  
};
```

Output parameter:

```
struct setSourceChannelPropertyResponse{  
    int result;           //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN

WS\_ERROR\_INVALID\_PARAM

WS\_ERROR\_CHANNEL\_NOT\_EXIST

Remarks:

Take effect after restart channel.

## 20. setMirrorChannelProperty

```
setMirrorChannelProperty(  
    char* pwd,           //access password  
    char* name,          //channel name  
    struct mirror_channel_property* in, //input, see input parameter for detail  
    struct setMirrorChannelPropertyResponse& out //output, see output parameter for  
    detail  
);
```

Input parameter:

```
struct mirror_channel_property{  
    int mdc;           //max directly connections.  
    int buffer;        //buffer size, in seconds.  
};
```

Output parameter:

```
struct setMirrorChannelPropertyResponse{  
    int result;           //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN  
WS\_ERROR\_INVALID\_PARAM  
WS\_ERROR\_CHANNEL\_NOT\_EXIST

Remarks:

Take effect after restart channel.

## 21. getLiveServerInfo

```
getLiveServerInfo(  
    char* pwd, //access password  
    struct getLiveServerInfoResponse& out //output, see output parameter for detail  
);
```

Output parameter:

```
struct license_info{  
    char* user_id; //User ID  
    char* user_name; //User Name  
    char* install_date; //License install date, format: 2010-09-09 08:00:00  
    int days; //can use days.  
    char* host_id; //HostID  
    char* desc; //Description  
};
```

```
struct live_server_info{  
    char* nat; //Nat server  
    char* cgi; //Tracker server  
    char* version; //  
    char* media_root_dir; //  
    struct license_info li; //  
};
```

```
struct getLiveServerInfoResponse{  
    struct live_server_info si; //  
    int result; //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS  
WS\_ERROR\_INVALID\_PWD  
WS\_ERROR\_UNKOWN

## 22. getLiveServerStatus

```
getLiveServerStatus(  
    char* pwd, //access password
```

```
struct getLiveServerStatusResponse& out //output, see output parameter for  
detail  
);
```

Output parameter:

```
struct live_server_status{  
    int online; //  
    int source_channels; //  
    int mirror_channels; //  
    char* start_dtime; //start time, format: 2010-09-09 08:00:00  
    int howlong_started; //seconds  
    double upload_speed; // KB/S  
    double total_upload; // MB  
    double download_speed; // KB/S  
    double total_download; // MB  
    int cpu_usage; // 0~100  
    int memory_usage; // MB  
};
```

```
struct getLiveServerStatusResponse{  
    struct live_server_status ss; //  
    int result; //return value, see return values for detail  
};
```

Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN

## 23. getRuntimeLog

```
getRuntimeLog(  
    char* pwd, //access password  
    struct getRuntimeLogIn* in, //input, see input parameter for detail  
    struct getRuntimeLogOut& out //output, see output parameter for detail  
);
```

Functional Description:

Get access log or status log from server, please find the detailed log format in the appendix.

Input parameter:

```
struct getRuntimeLogIn{  
    int nType; //1=access log, 2=status log  
    char* begin_date; // eg:2010-09-01
```

```
char* end_date;    // eg:2010-09-02
int nZipType;      //0=nozip,1=zipped use zlib.
};
```

Output parameter:

```
struct getRuntimeLogOut{
    struct xsd__base64Binary dataWithZip;    //zipped Log data.
    char* dataNoZip;                        //unzipped log data.
    int result;                             //return value, see return values for detail
};
```

Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN

WS\_ERROR\_NO\_RUNTIME\_LOG

#### 24. getSystemFilesList

```
getSystemFilesList(
    char* pwd,          //access password
    char* dir_path,     //relative path to the media_root_dir, eg:/movie.
                      //or the server absolute path.
    struct getSystemFilesListResponse& out //output, see output parameter for
detail
);
```

Output parameter:

```
struct ns__system_file{
    bool bDir;          //
    unsigned long long size; //in byte
    char* time;         //last modify time
    char* name;         //
};
```

```
struct system_file_list{
    struct ns__system_file* __ptr;
    int __size;
};
```

```
struct getSystemFilesListResponse{
    struct system_file_list sfl; //
    int result;                  //return value, see return values for detail
};
```



Return values:

WS\_SUCCESS

WS\_ERROR\_INVALID\_PWD

WS\_ERROR\_UNKOWN

## Appendix 1 Access Control

Access Control can protect the content published by the ISP.

There are 3 types of Access Control:

1. Simple password access control, the player will ask user to input the password. The ISP should set the password in the manage terminal.
2. User+Password access control, the player will ask user to input user and password, then post the user and password to the web page which you have set to do check. The ISP should set the URL of the user and password check web page in the manage terminal.(eg:http://www.xxx.com/usercheck.jsp) .

ISP should develop the web check page according the following format:

Player use "HTTP POST" to post the user and password which user input to the web page.

The content of the post is (UTF-8 encoding):

user=...&pass=...

pass is 32 uppercase characters of MD5.

The web page should return the following result by plain/text (UTF-8 encoding):

<result>

<login>success|fail</login>

<reason>...</reason>

</result>

If return fail, the player will display the text in the reason tag.

3. Domain access control, the player will read the URL of the current page, then post the access URL to the web page which you have set to do check. The ISP should set the URL of the domain check web page in the manage terminal.(eg:http://www.xxx.com/domaincheck.jsp).

ISP should develop the web check page according the following format:

Player use "HTTP POST" to post the access urlto the web page,.

The content of the post is (UTF-8 encoding):

url=...

url is the current access page (eg: <http://www.zz.com/play.jsp?id=123>).

The web page should return the following result by plain/text (UTF-8 encoding):

```
<result>
<login>success|fail</login>
<reason>...</reason>
</result>
```

If return fail, the player will display the text in the reason tag.

An jsp example:

```
<%@      page      language="java"      import="java.util.*"
pageEncoding="UTF-8"%>
<%
String url = request.getParameter("url");
String result = "";
String reason = "";
if (url != null && url.contains("www.nagasoft.cn")) {
    result = "success";
} else {
    result = "fail";
    reason = "please access from www.nagasoft.cn";
}
%>
<result>
<login><%=result%></login>
<reason><%=reason%></reason>
</result>
```

## Appendix 2 server log format

There are two type log, access log and status log. Each type log saved in one directory, one file for each day, use the date for the file name (eg: 2009-02-24), one line for each log record.

Live server access log format:

play: Date Time PeerId 1 "user ip address" "channel id" "channel name""connect directly to server"(1|0)

close: Date Time PeerId 1

eg:

2009-03-13 10:37:15 00E04D456BE7@5216QHY 1 125.31.197.217 1 ch1 1

2009-03-13 10:50:30 00E04D456BE7@5216QHY 0

Live server status log format:

Channel:Data Time 1 "Channel Id" "channel name" online "client directly connect to server" MDC "upload speed"(mb/s) "total uploaded"(MB) "download speed"(mb/s) "total downloaded"(MB)

Server:Date Time 0 online "upload speed"(mb/s) "total uploaded"(MB) "download speed"(mb/s) "total downloaded"(MB) "cpu usage"("%) "memory usage"(MB)

eg:

2009-03-13 10:37:15 1 2 ch2 123 50 50 80.123 23 12.456 13

2009-03-13 10:37:15 0 200 130.234 1000 20.123 120 11 300