

VJVOD3 SDK

VJVOD3 provide webservice interface, webservice description address:
<http://nagasoft.cn/download/ws/vjvod3/VJVodService.wsdl>

Document Conventions

1. String use UTF-8 encoding.
2. Webservice access password support plain text or md5 text (32 lowercase characters).
3. File HASH ID, the HASH ID is the same for the same file (no matter the file name or path, the file extension is the exception), eg: 2583402a56451b152a13ff9dc4c2a38e .
4. File HASH Code, including HASH ID, bitrate, duration, file size. eg:
hash=2583402a56451b152a13ff9dc4c2a38e&mime=flv&bitrate=209000&size=867306&time=32

Structure List

```
//string array
struct string_array{
    char** __ptr;
    int __size;
};

//int array
struct int_array{
    int* __ptr;
    int __size;
};

//VOD file info
struct ns__vod_file{
    char* path;      //file absolute path
    char* code;      //file hash code
};

//VOD file list
struct vod_file_list{
    struct ns__vod_file* __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
WS_ERROR_READ_FILE_INFO_FAILED	-30
WS_ERROR_FILE_PATH_EXIST	-31
WS_ERROR_FILE_HASH_EXIST	-32
WS_ERROR_INSERT_FILE_LIST_FAILED	-33
WS_ERROR_FILE_ZERO_SIZE	-34
WS_ERROR_FILE_NOT_IN_LIST	-35
WS_ERROR_FILE_DELETE_DELAYED	-36

Function List

1. publishFile

```
publishFile(  
    char* pwd,                //access password  
    char* file_path,          //absolute file path, eg: f:/media/123.rm  
    struct publishFileResponse& out //output, see output parameter for detail  
);
```

Output parameter:

```
struct publishFileResponse{  
    char* hash_code;          //file hash code  
    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_PATH_NOT_EXIST
WS_ERROR_READ_FILE_INFO_FAILED
WS_ERROR_FILE_ZERO_SIZE
WS_ERROR_FILE_PATH_EXIST
WS_ERROR_FILE_HASH_EXIST
WS_ERROR_INSERT_FILE_LIST_FAILED

Remarks:

File path and file hash id is unique, file with the same file path or hash id can't republished.

2. publishDir

```
publishDir(  
    char* pwd,                //access password  
    char* dir_path,           //absolute directory path  
    struct publishDirResponse& out //output, see output parameter for detail  
);
```

Functional Description

This function will Recursive search the directory and sub-directory, publish all the supported files.

Output parameter:

```
struct publishDirResponse{  
    struct vod_file_list vfl; //file list  
    int result;               //return value, see return values for detail  
};
```

Return values:

WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM

3. publishFileList

```
publishFileList(  
    char* pwd,                //access password  
    struct string_array* paths, //absolute file path array  
    struct publishFileListResponse& out //output, see output parameter for detail  
);
```

```
);
```

Output parameter:

```
struct publishFileListResponse{
    struct string_array hashcodes;    //file hash code array.
    struct int_array errors;          //error array.
    int result;                       //return value, see return values for detail
};
```

Return values:

WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM

4. unpublishFile

```
unpublishFile(
    char* pwd,                        //access password
    char* file_path,                  //absolute file path, eg:f:/media/123.rm
    struct unpublishFileResponse& out //output, see output parameter for detail
);
```

Functional Description

The vod file will can't play after success unpublished.

Output parameter:

```
struct unpublishFileResponse{
    char* hash_code;    //file hash code
    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_FILE_NOT_IN_LIST

5. unpublishFileByHash

```
unpublishFileByHash(
    char* pwd,                        //access password
    char* hash,                       //file hash id
    struct unpublishFileByHashResponse& out //output, see output parameter for detail
);
```

```
);
```

Output parameter:

```
struct unpublshFileByHashResponse{
    char* file_path;           //absolute file path
    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_FILE_NOT_IN_LIST

6. unpublshDir

```
unpublshDir(
    char* pwd,                 //access password
    char* dir_path,            //absolute directory path
    struct unpublshDirResponse& out //output, see output parameter for
detail
);
```

Functional Description

This function will Recursive search the directory and sub-directory, unpublsh all the supported files.

Output parameter:

```
struct unpublshDirResponse{
    struct vod_file_list vfl; //file list
    int result;                //return value, see return values for detail
};
```

Return values:

WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM

7. unpublshFileList

```
unpublshFileList(
    char* pwd,                 //access password
    struct string_array* paths, //absolute file path array.
    struct unpublshFileListResponse& out //output, see output parameter for detail
);
```

```
);
```

Output parameter:

```
struct unpublishFileListResponse{  
    struct string_array hashcodes;           //file hash code array.  
    struct int_array errors;                 //error code array.  
    int result;                             //return value, see return values for detail  
};
```

Return values:

WS_SUCCESS

WS_ERROR_INVALID_PWD

WS_ERROR_UNKOWN

WS_ERROR_INVALID_PARAM

8. deleteFile

```
deleteFile(  
    char* pwd,                             //access password  
    char* path,                             //absolute file path  
    struct deleteFileResponse& out         //output, see output parameter for  
    detail  
);
```

Functional Description

Unpublish the file and delete it from the hard disk, the server will retry if the file is busy.

Output parameter:

```
struct deleteFileResponse{  
    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_FILE_DELETE_DELAYED

9. deleteFileByHash

```
deleteFileByHash(  
    char* pwd,                             //access password  
    char* hash,                             //file hash id  
    struct deleteFileByHashResponse& out    //output, see output parameter for detail  
);
```

```
);
```

Functional Description

The server will find the file according to HASH ID, then unpublish the file and delete it from the hard disk, the server will retry if the file is busy.

Output parameter:

```
struct deleteFileByHashResponse{
    char* file_path;           //file path
    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_FILE_NOT_IN_LIST
WS_ERROR_FILE_DELETE_DELAYED

10. deleteDir

```
deleteDir(
    char* pwd,                 //access password
    char* dir_path,            //absolute directory path
    struct deleteDirResponse& out //output, see output parameter for detail
);
```

Functional Description

This function will Recursive search the directory and sub-directory, unpublish all the supported files, and then delete the directory or files from the hard disk, the server will retry if the file is busy.

Output parameter:

```
struct deleteDirResponse{
    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_FILE_DELETE_DELAYED

11. deleteFileList

```
deleteFileList(  
    char* pwd,                //access password  
    struct string_array* paths, //file path array.  
    struct deleteFileListResponse& out //output, see output parameter for  
    detail  
);
```

Functional Description

This function will unpublish all the files in the input list, and then delete all the files from the hard disk, the server will retry if the file is busy.

Output parameter:

```
struct deleteFileListResponse{  
    struct int_array errors; //error array  
    int result;              //return value, see return values for detail  
};
```

Return values:

WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM

12. getFileList

```
getFileList(  
    char* pwd,                //access password  
    int offset,               //<=0 from begin, take effect when number>0  
    int number,               // <=0 means all the files.  
    struct getFileListResponse& out //output, see output parameter for detail  
);
```

Output parameter:

```
struct getFileListResponse{  
    struct vod_file_list vfl; //file list  
    int result;               //return value, see return values for  
    detail  
};
```

Return values:

WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN

13. searchFileByName

```
searchFileByName(  
    char* pwd,                //access password  
    char* namekey,            //search key  
    struct searchFileByNameResponse& out //output, see output parameter for  
    detail  
);
```

Output parameter:

```
struct searchFileByNameResponse{  
    struct vod_file_list vfl;    //file list  
    int result;                  //return value, see return values for  
    detail  
};
```

Return values:

WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM

14. searchFileByHash

```
searchFileByHash(  
    char* pwd,                //access password  
    char* hash,               //HASH ID  
    struct searchFileByHashResponse& out //output, see output parameter for  
    detail  
);
```

Output parameter:

```
struct searchFileByHashResponse{  
    struct vod_file_list vfl;    //file list  
    int result;                  //return value, see return values for  
    detail  
};
```

Return values:

WS_SUCCESS
WS_ERROR_INVALID_PWD
WS_ERROR_UNKOWN
WS_ERROR_INVALID_PARAM

15. getServerStatus

```
getServerStatus(  

```

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

Output parameter:

```
struct server_status{
    int cid; //channel id
    int online; //
    int files_num; //the count of published files
    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 getServerStatusResponse{
    struct server_status ss; //
    int result; //return value, see return values for detail
};
```

Return values:

WS_SUCCESS

WS_ERROR_INVALID_PWD

WS_ERROR_UNKOWN

16. setAccessCheck

```
setAccessCheck(
    char* pwd, //access password
    struct access_check* in, //Input, see input parameter for detail
    struct setAccessCheckResponse& 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 setAccessCheckResponse{
    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_TRACKER_REPORT_FAILED

17. getAccessCheck

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

Output parameter:

```
struct getAccessCheckResponse{
    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

18. setAD

```
setAD(
    char* pwd,          //access password
    char* ad,          // read from vva file. Use “VJADMaker” to create vva files.
    struct setADResponse& out //output, see output parameter for detail
);
```

Output parameter:

```
struct setADResponse{  
    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_TRACKER_REPORT_FAILED
```

19. getAD

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

Output parameter:

```
struct getADResponse{  
    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
```

20. getServerInfo

```
getServerInfo(  
    char* pwd,          //access password  
    struct getServerInfoResponse& 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 time, eg:2010-09-09 08:00:00  
    int days;          //can used days  
    char* host_id;      //Host id  
    char* desc;         //Description  
};
```

```
struct server_info{
    char* nat;           //Nat server
    char* cgi;           //Tracker server
    char* version;       //version
    char* media_root_dir; //media root dir
    int cid;             //channel id
    struct license_info li; //license info
};

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

Return values:

WS_SUCCESS

WS_ERROR_INVALID_PWD

WS_ERROR_UNKOWN

21. 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

22. 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.

Vod Server access log format:

Play: Date Time PeerId 1 "user ip address" "file hash id" "file path" success(1|0)

Close: Date Time PeerId 0

eg:

```
2009-03-13 10:37:15 00E04D456BE7@5216QHY 1 125.31.197.217
b1d76323a625f34fd12d871597eadcbf "f:\media\test.wmv" 1
2009-03-13 10:50:30 00E04D456BE7@5216QHY 0
```

Vod server status log format:

Date Time Online "upload speed"(mb/s) "total uploaded"(MB) "download

speed"(mb/s) "total downloaded"(MB) "cpu usage"(%) "memory usage"(MB)

eg:

2009-03-13 10:37:15 123 120.431 1023 10.123 800 12 450