API PRODUCT DOCUMENTATION

Barcelona, August 2014, v1.1.3

Marc Maycas Claramunt
marc@nicepeopleatwork.com
C/ Doctor Trueta 127-133
08005 Barcelona

Version	Date	Description	Author
1.0.0	02/05/2013	Initial draft	Marc Maycas
1.1.0	06/05/2013	Added and corrected information	Marc Maycas
1.1.1	17/04/2014	Updated license URL	Marc Maycas
1.1.2	05/08/2014	Examples and resources	Marc Maycas

Table of Contents

General

About this document

After reading this document, you should have all the necessary information to set up and operate NicePeopleAtWork's Transcoder and Multi-DRM packager using different Nice Packager API calls. Therefore, it will be possible to transcode, encrypt and package Video On Demand (VoD) assets and Live streams with any of the currently supported third party DRM technologies: *Google/Widevine*, *Adobe Flash Access, Microsoft PlayReady* and *Windows Media DRM*[1]

This way, there won't be any need to log into an FTP server to upload an XML job file or log into the packager user interface to check its status.

Nice People At Work

Nice People At Work is a fast-growing company based in Barcelona (Spain) that provides comprehensive yet cost efficient Internet Video Distribution solutions. We create audio and video distribution technologies to meet the requirements of premium content owners, broadcasters and telcos, so they can provide the best experience across all devices.

We create solutions that enable premium video experiences in existing media workflows. We have gained a lot of know-how in dealing with the most premium corporate and consumer video (from live soccer to Hollywood movies). Within our service offering we support most DRM (Digital Rights Management) technologies such as Widevine, PlayReady, Adobe Flash Access, Marlin and Windows Media DRM so we can reach any device in the market with the best possible experience.

Through our deep involvement in solving our customers' complex video distribution problems, we have gained great understanding in everything required to launch and operate a successful video-centric Internet-based operation.

Our solutions include content adaptation (transcoding and DRM protection), secure content delivert and QoS (monitoring, live analytics and real time CDN load balancing). We serve customers around the world, enabling their offering for premium video experiences.

[1] At the time of this writing, Intertrust Marlin is also supported by Nice People At Work for VoD. However, this DRM technology is integrated in its OVP instead of inside the Nice Packager. Further product updates may include it.

Nice Packager REST API Reference

A basic structure of an API call is as follows:

```
http://[AZURE DNS]:8182?<GET_PARAM1>&<GET_PARAM2>&...
```

where `GET_PARAM` are the different options or parameters that will be passed through a GET Method.

Hereunder you can see the list and description of all the available methods:

1. Add a new job

Description	Creates a new job for the packager to process
GET Parameters	➢ `operation=newjob`
POST Parameters	➢ XML or JSON job data[2]

Response messages (in JSON):

Correct Operation

Depending on if a JSON or an XML file has been uploaded, the following response messages will be received:

```
{"status":"OK","jobhash":"JOB_HASH_ID","message":"XML Received"}
```

```
{"status":"OK","jobhash":"JOB_HASH_ID","message":"JSON Received"}
```

where:

- `JOB_HASH_ID` is the job identifier inside Nice Packager

Error Operation

```
{"status":"KO","message":"ERROR_MESSAGE"}
```

List of ERROR_MESSAGE:

```
- Operation unknown
- Operation undefined
- Post message is empty
- Source tag is undefined
- Result tag is undefined
- Job doesn't have any task
- Invalid XML format
- Invalid JSON format
```

[2] The job structure is defined in the next section

- XML couldn't be generated

2. Show status

Description	Shows a job status if given a job ID or all the jobs in case the job ID is not specified.
GET Parameters	➢ `operation=showstatus` (mandatory) ➢ `job=JOB_HASH_ID` (optional)
Considerations	If the `job` parameter is not specified, the packager will return all the jobs.

Response messages (in JSON):

Correct Operation

```
{"status":"OK","jobs":[JOB_INFO_1,JOB_INFO_2,...,JOB_INFO_N]}
```

where:

- `JOB_INFO_X` is:

```
{"job":"JOB_HASH_ID","name":"JOB_NAME","assetName":"ASSETNAME","created_at":"dd/MM/yyyy HH:mm:ss","saveTranscoding":"true/false"","priority":PRIORITY_INT,"transcoding":[TASK_TRANS_1,...,TASK_TRANS_M],"packaging":[TASK_PACK_1,...,TASK_PACK_P]}
```

- `TASK_TRANS_X` is:

```
{"task":"transcoding","name":"TASK_NAME","status":"STATUS_VALUE","statustext":"STATUS_TEXT","percent":"PERCENT_TASK","errormsg":"ERROR_MESSAGE"}
```

List of STATUS_VALUE:

- Pending
- Complete
- Error
- Running
- Downloading
- Transcoding
- Uploading
- Cleaning trash

- `TASK_PACK_X` is:

```
{"task":"packaging","name":"TASK_NAME","type":"DRM_TYPE","status":"STATUS_VALUE","statustext":"STATUS_TEXT","percent":"PERCENT_TASK","errormsg":"ERROR_MESSAGE"}
```

List of STATUS_VALUE:

- Pending
- Complete
- Error
- Running
- Downloading
- Packaging
- Registering
- Getting keys
- Generating fragments
- Making config files
- Uploading files
- Cleaning trash

- ERROR_MESSAGE is an exception message launched by any of the involved processes.

MESSAGE EXAMPLE


```
{
  "status": "OK",
  "jobs": [
    {
      "job": "6e274fd8f59d220bfa540c9d9f9e9a4e01f9583c706d3bc6",
      "name": "Sintel A",
      "assetName": "leaseWebSintel1",
      "created_at": "20/09/2013 12:38:27",
      "saveTranscoding": "true",
      "priority": 7,
      "transcoding": [
        {
          "task": "transcoding",
          "name": "SS 16:9 404p 750 MP30",
          "status": "Downloading",
          "statustext": "1/4",
          "percent": 54,
          "errormsg": ""
        },
        {
           "task": "transcoding",
           "name": "SS 16:9 352p 1100 MP30",
           "status": "Downloading",
           "statustext": "1/4",
           "percent": 54,
           "errormsg": ""
        }
      ],
      "packaging": [
      {
        "task": "packaging",
        "name": "Widevine Trickplay SD",
        "type": "WIDEVINE",
        "status": "Downloading",
        "statustext": "1/5",
        "percent": 54,
        "errormsg": ""
      },
      {
        "task": "packaging",
```

```
          "name": "Video PlayReady",
          "type": "PLAYREADY",
          "status": "Downloading",
          "statustext": "1/5",
          "percent": 54,
          "errormsg": ""
        },
        {
          "task": "packaging",
          "name": "Video AFA",
          "type": "AFA",
          "status": "Downloading",
          "statustext": "1/5",
          "percent": 54,
          "errormsg": ""
        }
      ]
    }
  ]
}
```


Error Operation


```
{"status":"KO","message":"ERROR_MESSAGE"}
```


List of ERROR_MESSAGE:

```
- Operation unknown
- The list of jobs is empty
- No job to show
```

3. Job Cancellation

Description	Cancels an specific job
GET Parameters	➢ `operation=canceljob` (mandatory) ➢ `job=JOB_HASH_ID` (mandatory)
Considerations	At this version of the packager jobs must be cancelled one by one

Response messages (in JSON):

Correct Operation


```
{"status":"OK","deleted":DELETED_MESSAGE}
```


where:

- `DELETED_MESSAGE` is:
  - `true`: If the job has been successfully cancelled
  - `false`: If the job has already been cancelled

Error Operation


```
{"status":"KO","message":"ERROR_MESSAGE"}
```


List of ERROR_MESSAGE:

- Operation unknown
- Need the job ID to cancel it

4. Changing job priority

Description	Modifies a job priority
GET Parameters	➢ operation=changepriority (mandatory) ➢ job=JOB HASH ID (mandatory) ➢ priority=PRIORITY_VALUE (mandatory)
Considerations	The priority value must be between 1 and 9

Response messages (in JSON):

Correct Operation

```
{"status":"OK","modified":"true"}
```

Incomplete Operation

In case the operation can't be completed, 2 self- explanatory messages can appear

```
{"status":"OK","modified":"false","message":"Job is running now"}
```

```
{"status":"OK","modified":"false","message":"Job does not exist"}
```

Error Operation

```
{"status":"KO","message":"ERROR_MESSAGE"}
```

List of ERROR_MESSAGE:

- Operation unknown
- Need the job ID and priority to change it

Nice Packager Job Structure

Set up VoD Transcoding, Encryption and Packaging

VoD Transcoding, Encryption and Packaging XML job definition

In this section can be found the structure of a job, which is the XML file containing all the tasks that will be carried out (transcoding, encryption and packaging)

The XML file has the following structure[3]:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job>
   <name> </name>
   <priority> </priority>
   <description> </description>
   <assetName> </assetName>
   <accountCode> </accountCode>
   <source>
         <type>  </type>
         <host> </host>
         <path> </path>
         <user> </user>
         <password> </password>
         <accesskey> </accesskey>
         <secretkey> </secretkey>
         <bucketname> </bucketname>
          <regionname> </regionname>
   </source>
   <result>
         <type> </type>
         <host> </host>
         <path> </path>
         <user> </user>
         <password> </password>
          <accesskey> </accesskey>
          <secretkey> </secretkey>
           <bucketname> </bucketname>
           <regionname> </regionname>
   </result>
   <transcoding>
         <source> </source>
         <thumbs>
                <thumbsFolder> </thumbsFolder>
                <thumbsInterval> </thumbsInterval>
                <thumbsResolution> </thumbsResolution>
         </thumbs>
         <tasks>
                <task>
                       <name> </name>
                       <qualityKey> </qualityKey>
                       <resultFilename> </resultFilename>
                       <saveTranscoding> </saveTranscoding>
                       <encodingParams>
                              <height> </height>
                              <width> </width>
                              <videoBitrate> </videoBitrate>
                              <audioBitrate> </audioBitrate>
                              <gop> </gop>
                              <pass> </pass>
                              <extra> </extra>
                       </encodingParams>
                </task>
               {...}
```

[3] Take a look at the appendices section for some XML examples.

```
        </tasks>
   </transcoding>
   <packaging>
        <saveTranscoding> </saveTranscoding>
        <packages>
               <package>
                      <name>(package name)</name>
                      <type>(WIDEVINE, PLAYREADY,...)</type>
                      <resultFilename>resultFilename</resultFilename>
                      <trickPlayGop>Y/N</trickPlayGop>
                      <filesGop></filesGop>
                      <files>
                             <file>
                                    <filename>name of the resulting file</filename>
                                    <isTrickPlay>Y/N</isTrickPlay>
                             </file>
                             {...}
                      </files>
                </package>
        {...}
        </packages>
   </packaging>
   <notifications>
        <start> </start>
        <done> </done>
        <error> </error>
   </notifications>
</job>
```

As you can see in this XML structure, each task is included inside a <job> tag. Each job will correspond to a transcoding, encryption and packaging petition with the possibility to operate it in an "Only transcoding/packaging scenario" if necessary by not including the corresponding tag.

Going into detail with the XML, you can see that there are the following fields and subfields:

- **Name:**

  Job Name.

- **Priority:**

  Integer ([0, 100]) that specifies the priority of the process.

  If a process has a higher priority value than a queued one, it will be processed first.

  If two processes have the same priority value, they will be processed following a FIFO rule (First In, First Out).

- **Description:**

  Short description of the process.

- **AssetName:**

  File name that will be registered in the DRM provider systems (Adobe, Google, Microsoft) in case you are packaging to a specific DRM technology.

Important Note: Widevine does not register an asset name but an asset id. However, Google servers need an asset name as an input.

When packaging a Widevine file the packager will return an XML containing the asset ID in order to be used when retrieving the license for an specific content (you can see the XML structure of Widevine in the appendix section).

- **AccountCode:**

  Your Nice PeopleAtWork account code that indicates which customer account you are sending the jobs to. This parameter will be provided by NicePeopleAtWork. If you don't have it yet, please contact your Customer Engineer or Support Agent.

- **Source:**

  It defines where are located the files that will be encrypted and packaged. The following fields must be defined:

Parameter	Mandatory?	Description
Type	Yes (not case sensitive)	The following values are allowed: STD: The files are located inside the local FTP FTP: The files are located inside a remote FTP server SSH: The files are located inside a remote SSH server HTTP: The files are located inside a remote HTTP server S3: The files are located inside an Amazon S3 account.
Host	Only if type equals to FTP, SSH or HTTP	Host where the files are located
Path	Optional only in case of using S3 type. For other types is mandatory	Path where the files are located.
User	Only if type equals to FTP or SSH	Username to access to the remote server
Password	Only if type equals to FTP or SSH	Password to access to the remote server
Access key	Only if type equals to S3	Variable used by Amazon S3 so the packager can use the API
Secret key	Only if type equals to S3	Variable used by Amazon S3 so the packager can use the API
Bucket Name	Only if type equals to S3	Name of the storage area the user has defined in Amazon S3
Region Name[4]	Only if type equals to S3	Region defined by Amazon S3 where the files are located

- **Result:**

  It defines where the resulting files will be left after completing all the tasks and packagings:

[4] For more information regarding Amazon S3 Region Names, take a look at the following link for an updated information:

http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/regions/Regions.html

Parameter	Mandatory?	Description
Type	Yes (not case sensitive)	The following values are allowed: STD: The files will be placed in the local packager's FTP FTP: The files will be placed inside a remote FTP server SSH: The files will be placed inside a remote SSH server S3: The files will be placed inside an Amazon S3 account.
Host	Only if type equals to FTP, SSH or HTTP	Host where the files will be placed
Path	Optional only in case of using S3 type. For other types is mandatory	Path where the files will be placed
User	Only if type equals to FTP or SSH	Username to access to the remote server
Password	Only if type equals to FTP or SSH	Password to access to the remote server
Access key	Only if type equals to S3	Variable used by Amazon S3 so the packager can use the API
Secret key	Only if type equals to S3	Variable used by Amazon S3 so the packager can use the API
Bucket Name	Only if type equals to S3	Name of the storage area the user has defined in Amazon S3
Region Name	Only if type equals to S3	Region defined by Amazon S3 where the files will be located

- Transcoding (OPTIONAL):

Inside this block there are all the transcoding tasks to be done to a given master file.

The transcoding block is not mandatory. It can be omitted if the packager only needs to encrypt and package a group of files.

Inside, you will find the following tags:

Parameter	Mandatory?	Description
source	Yes	Source video filename included inside the previously defined source path
thumbs	Yes	It includes all the options regarding the video thumbnails
tasks	Yes	All the information of that transcoding task.

- Thumbs:

A thumbnail is a reduced size version of a picture that is used to help in recognizing and organizing the larger scale ones, serving the same role for images as a normal text index does for words.

Inside the thumbs tag, you can define the following:

Parameter	Mandatory?	Description
thumbsFolder	No	The path where the thumbnails will be stored. If this parameter is not informed, then the result will be stored in: resultPath/thumbs
thumbsInterval	No	Interval between generated thumbs in seconds

thumbsResolution	No	Resolution of the generated thumbnails. If not informed, the resolution is the same as the video input.

- Tasks:

Inside this block, a list of different tasks can be defined.

Each task is a single quality file to be created from the video master file.

It includes:

Parameter	Mandatory?	Description
name	Yes (if <transcoding> is included)	Task name
qualityKey	Yes (if <transcoding> is included)	Quality to be created. It defines all the encoding parameters for that quality. These quality codes are the same than the ones used in NicePeopleAtWork's Premium Video Platform[5]. By doing this, there is no need to define new encoding profiles.
resultFilename	Yes (if <transcoding> is included)	Name of the resulting file
saveTranscoding	No	If set to true, it lets you save and upload to your CDN Origin the resulting encoded file. The value of this field prevails over the saveTranscoding value in Packaging.
encodingParams	No	Nice Packager also allows to overwrite some of the encoding parameters defined by the qualityKey. However the only available codecs are H264 for video and AAC for audio. By doing this, all connected devices can be reached.

- Encoding Parameters:

As described, each qualityKey defines the encoding parameters for each file. However, some parameters can be redefined by changing the information comprised in the following tags:

Parameter	Mandatory?	Description
height	No	Specifies the transcoded file image height. If nothing is specified it takes the original file height.
width	No	Specifies the transcoded file image width. If nothing is specified it takes the original file width.
videoBitrate	No	Transcoded video's video bitrate
audioBitrate	No	Transcoded video's audio bitrate
gop	No	Group of Pictures for the resulting file[6]
pass	No	If the value is equal to: 1 - Single encoding pass 2 - Double encoding pass
extra	No	If you want to add extra fmpeg encoding parameters introduce them in one line between this tag[7]

- Packaging (OPTIONAL):

Inside this block there all the packaging tasks to be done to a transcoded file or a group of transcoded files.

[5] See in the appendix the complete list for qualityKeys
[6] For more information on this parameter, take a look at: http://en.wikipedia.org/wiki/Group_of_pictures
[7] For more information on ffmpeg flags, take a look at: http://ffmpeg.org/ffmpeg.html

Like the transcoding block, the packaging block is not mandatory. It can be omitted if the packager only needs to transcode a file.

Inside, you will find the following tags:

Parameter	Mandatory?	Description
saveTranscoding	Yes	If set to true, it lets you save and upload to your CDN Origin the previously encoded or imported files to be encrypted and packaged.
packages	Yes (if <packaging> is included)	It includes a list of different <package> tags. For each DRM package to be generated, a new <package> tag will need to be created.

- Packages:

Inside packages, you will have to generate a different set of <package> blocks that will contain the following parameters:

Parameter	Mandatory?	Description
name	Yes (if <packaging> is included)	Package name
type	Yes (if <packaging> is included)	It defines the DRM technologies that will be applied to the previous source files. The allowed values are the following (no case sensitive): - "widevine" (for Google Widevine) - "playready" (for Microsoft PlayReady) - "afa" (for Adobe Flash Access) Just a single type is allowed per package, so different packaged files will require new <package> blocks
resultFilename	Yes (if <packaging> is included)	Resulting packaged file name
trickPlayGop	Only for Widevine. For other DRMs is optional (if <packaging> is included)	If there's a trickplay file[8] to be used, this number that indicates which GOP (Group of Pictures) has. This field only applies if encrypting to **Google Widevine**.
filesGop	Only for Widevine. For other DRMs is optional (if <packaging> is included)	GOP of the files that will compose the resulting packaged file. This field only applies if encrypting to **Google Widevine**.
files	Yes (if <packaging> is included)	Defines the list of files that will be included inside a packaged file.

- **Files:**

Inside Files, the following paramters must be defined:

Parameter	Mandatory?	Description
filename	Yes (case sensitive)	File name
isTrickPlay	Optional. Only with Widevine and if there's a trickplay file	The following values are allowed: Y: Yes. If the file is a trickplay file (only one must have this value active) N: No

[8] trickPlay is the feature that lets you to rewind or fast forward the video. The file used for trickPlay is always the file with the lowes bitrate

As shown in the XML structure, the fields `filename` or `isTrickPay` have to be repeated as many times as files you have to package.

- **Notifications (OPTIONAL):**

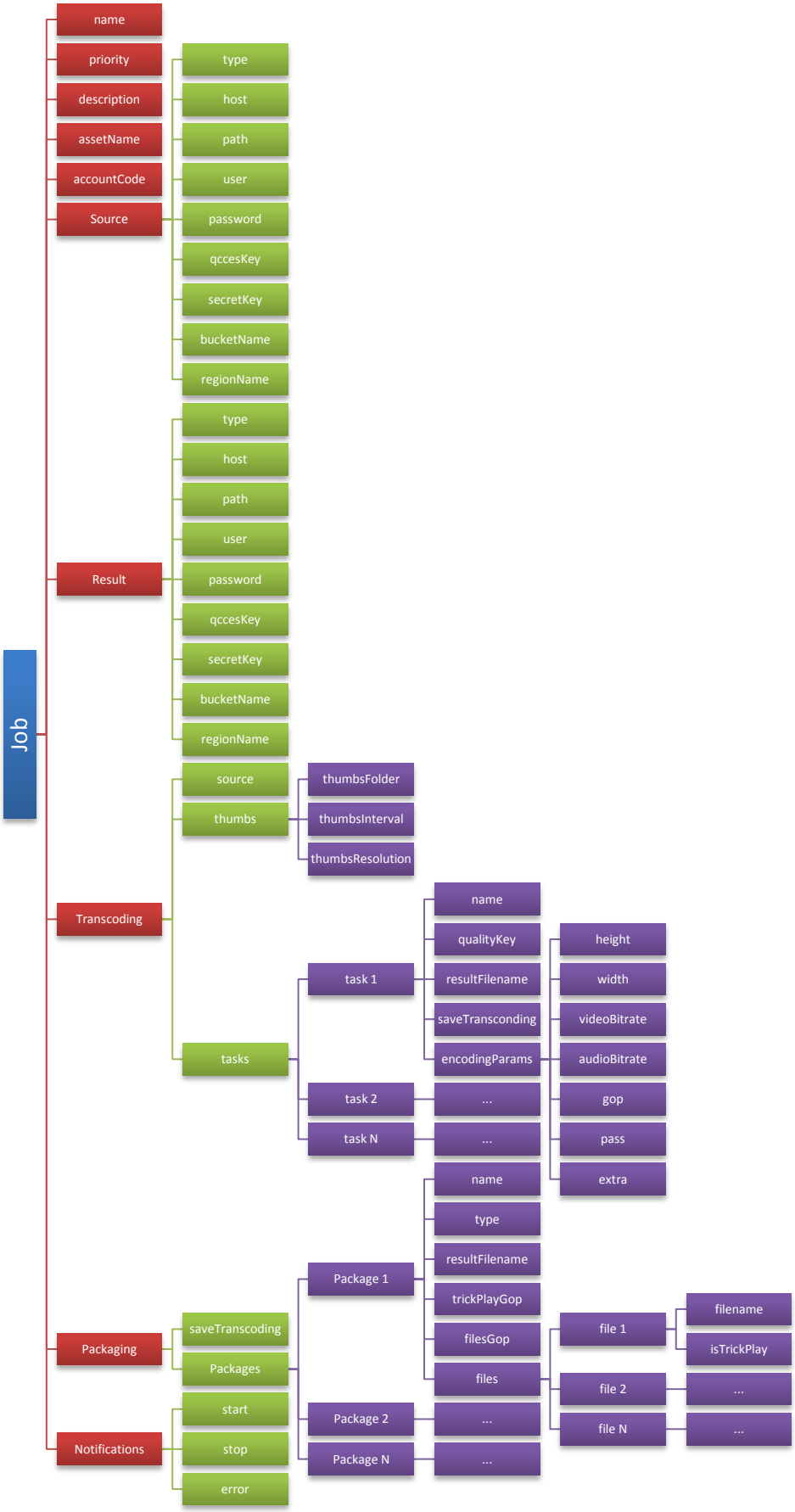
Inside this block, there will be defined all the customer webservice URLs where the Nice Packager will send notifications to. The webservice is maintained by the customer

It has the following tags inside:

Parameter	Mandatory?	Description
start	No	The webservice URL where a notification will be sent once a new job starts
done	No	The webservice URL where a notification will be sent once a new job ends correctly
error	No	The webservice URL where a notification will be sent if an error occurs

The message detail the Nice Packager sends to the customer webservice is described in the Appendix B of this documentation.

To recap all the information, please find below a diagram with the entire described job XML structure organized in a more logical way:


Job
name
priority
description
assetName
accountCode
Source
type
host
path
user
password
qccesKey
secretKey
bucketName
regionName
Result
type
host
path
user
password
qccesKey
secretKey
bucketName
regionName
Transcoding
source
thumbs
thumbsFolder
thumbsInterval
thumbsResolution
tasks
task 1
task 2
task N
name
qualityKey
resultFilename
saveTransconding
encodingParams
...
...
height
width
videoBitrate
audioBitrate
gop
pass
extra
Packaging
saveTranscoding
Packages
Package 1
Package 2
Package N
name
type
resultFilename
trickPlayGop
filesGop
files
...
...
file 1
file 2
file N
filename
isTrickPlay
...
...
Notifications
start
stop
error

Nice Packager notification callbacks

NicePackager can send notification callbacks of a specified job a customer's webservice.

As seen in the XML definition, 3 different callback events can be sent to given webservice URLs: Start, Done, Error.

It's up to the customer to develop the webservice (or webservices) that will receive the following messages.

1. START job callback

```
{"notification":"start","jobhash":"JOB_HASH_ID"}
```

2. DONE job callback

Job completed successfully:

```
{"notification":"done","jobhash":"JOB_HASH_ID","message":"Job complete"}
```

Job completed with errors:

This status occurs when a job has ended but one or many tasks have errors.

```
{"notification":"done","jobhash":"JOB_HASH_ID","message":"Job completed with
errors"}
```

3. ERROR job callback

```
{"notification":"error","jobhash":"JOB_HASH_ID","message":"ERROR_MESSAGE"}
```

List of ERROR_MESSAGE:

DRM technology not activated messages

- *Packager doesn't support Widevine*
- *Packager doesn't support PlayReady*
- *Packager doesn't support Adobe Flash Access*

Result server error messages

- *Need info for FTP Result*
- *Need info for SSH Result*
- *Need info for S3 Result*
- *Need info for STD Result*
- *Undefined Result Type*

Source server error messages

- *Need info for FTP Source*
- *Need info for SSH Source*
- *Need info for S3 Source*
- *Need info for HTTP Source*
- *Need info for STD Source*
- *Undefined Source Type*

DRM related error messages

- *Widevine error registering*
- *Widevine error packaging*
- *FlashAccess error in the generation of fragments files*
- *FlashAccess error uploading video fragments*
- *Playready error getting info from Key Server*
- *Playready error in the generation of fragments files*
- *Playready error uploading video fragments*

Other operational messages

- *The system can't create directory: NAME_DIRECTORY*
- *Exception Message: EXCEPTION_MESSAGE*
- *Error downloading file: NAME_FILE*
- *Error transcoding*
- *Error uploading result file: NAME_FILE*
- *Error generating xml result file: NAME_FILE*

Nice Licenses

In this section of the documentation, it is explained how to generate the necessary license URL to retrieve the content license and be able to play the encrypted video.

Unlike the Nice Packager, Nice Licenses doesn't have a GUI and all the parameters (business rules, security tokens...) are added inside the request URL.

The service URL to request a license is: `http://drm.license.nice264.com/`

URL Parameters

Hereunder there's a list with all the parameters that can be added in the request URL and its purpose. Not all of them are mandatory, but they unlock the full potential of the platform with extra features that premium content providers will find useful.

Parameter	Mandatory?	Type	Description
license	Yes (even if it's generated in blank)	urlencoded string from a previous base64 encoded string	Parameter which contains the business rules
ip	Optional	String	IP address of the end user requesting the license
contentId	Yes	String	Content identifier that will be played
mediaId	Optional	String	Content hash if the video is hosted in NicePeopleAtWork infrastructure. This parameter will also appear in Nice Analytics to be able to track it
transactionId	Optional	String	Identifier used to moniror the transaction from NicePeopleAtWork's Analytics service
transactionTime	Optional	Long	Moment (in milliseconds) at which the transaction begins. Only applicable if the transactionId has been defined
accountCode	Yes	String	The identifier or code of the client through which the client request is made. It is used for security and to verify the authenticity of the token. This parameter will be provided by your sales engineer
expireTime	Yes	Long	It defines the moment at which the license expiresand from which it will no longer grant access to the content
niceToken	Yes	String	MD5 Token authenticator to validate if the license request is correct

Generating the 'license'

The *license* parameter contains the business rules that will be used to regulate the encrypted content playback.

As mentioned before, license contains a URL encoded string from a previous base 64 encoded string, which is a concatenation of all the business rules parameters with ampersands (&) and each value composed as 'key=value'.

The list of parameters defined in license is:

Parameter	Type	Description
duration	uint32	Number of seconds during which the generated authorisation URL is valid
purchaseDuration	uint32	The duration of the purchase in seconds. Commonly known as the rent window or validity window of the license (24h, 48h…). This value must be different than 0 and generally higher than the 'duration' parameter
countries	string csv (comma separated values)	Country list (listed using code ISO 3166-1 alpha-2), separated by commas, in which the video can be played. If this field is empty, then the video can be played in all countries (no geoblocking). Example: countries = "ES, DE, BO"

Note that all these parameters are optional. The customer can enable the playback without adding them in the license. **However**, the license is a mandatory parameter to be included in the request URL even if it's with all its parameters in blank.

Thus, Nice License will receive a license formatted like:

duration=100&purcharseDuration=300&countries=ES,AR

Remember, that when adding this parameter it has to be encoded in Base64 and, afterwards, with URLencode[9].

Generating the 'niceToken'

The niceToken sent as a parameter in the URL request has to be created beforehand. For this, the niceToken uses a concatenation of the previously described parameters with a couple of extra ones, as it is described with the following expression:

ruleParam1 + *license*[10] + ip + contentId + mediaId + transactionId + transactionTime + accountCode + expireTime + ruleParam2

You will see that there are two different parameters called ruleParam1 and ruleParam2, which are secret keys provided by your NicePeopleAtWork's sales engineer and unique for each NicePeopleAtWork account.

Finally, this concatenated string has to be encoded using MD5 to obtain the result to pass as a parameter.

Determining the right DRM technology

Once the right license URL is created and validated[11], the DRM technology type that is provided to the player comes from the parameters that the player also sends when requesting the license.

This is why no 'DRM' parameter is attached to the DRM request.

[9] See the PHP example in the appendix to understand how to format it

[10] The value of the license text string will be as received by the proxy. In other words, it will be transformed into Base64, but not encoded with URLEncode.

[11] See next section 'Nice License Security Layer' to see the validation and security rules the License proxy executes in order to check if an incoming license request is valid or not.

Nice License Security Layer

The License proxy goes through an initial validation process to ensure that the type of license is correct. In this section, the four stages of validation are described to provide information on which security validations are done when sending a license request:

Valid IP

The Nice License proxy stores the IP of the incoming request and compares it against the 'ip' that is sent as a parameter. If the values are identical, it proceeds with the validation process; otherwise the license will not be served.

Because the 'ip' parameter is optional, if it is not sent, then this verification won't take place.

Expire Time

The proxy will verify that the expireTime that has been sent as a parameter is greater than the value of the current time. In other words, it will check whether the license request has expired or not.

If it has not expired, the validation process will continue, otherwise the license will not be served.

Valid Geo IP

The third step is to verify that the license allows video playback in the country from which the request has been made. To do this, the proxy determines the countries in which the video can be played, based on the business rules, and will check that the user’s country has that access enabled. Otherwise, the license won't work.

Because the definition of the enabled countries is optional, this verification will not always take place.

Valid Token

During the last step, the proxy will verify that the token received is correct, by creating a token with the received parameters and checking it against the one that has been received.

If the token is valid, all the validations will have passed and a license will be served.

Appendix

Nice Packager VoD XML example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job>
    <name>Widevine and Playready SD</name>
    <priority>9</priority>
    <description>description</description>
    <assetName>exampleassetname1</assetName>
    <accountCode>nicetv</accountCode>
    <source>
        <type>std</type>
        <host></host>
        <path></path>
        <user></user>
        <password></password>
    </source>
    <result>
        <type>ftp</type>
        <host>46.165.236.139</host>
        <path>/home/FTP-shared/upload/</path>
        <user>userftp</user>
        <password>userftp</password>
    </result>
    <transcoding>
        <source>clipcanvas.mp4</source>
        <thumbs>
                <thumbsFolder></thumbsFolder>
                <thumbsInterval></thumbsInterval>
                <thumbsResolution></thumbsResolution>
        </thumbs>
        <tasks>
                <task>
                        <name>SS 16:9 404p 750 MP30</name>
                        <qualityKey>main_1300k_std_afa_24FPS</qualityKey>
                        <resultFilename>video1.mp4</resultFilename>
                        <encodingParams>
                                <height></height>
                                <width></width>
                                <videoBitrate></videoBitrate>
                                <audioBitrate></audioBitrate>
                                <gop></gop>
                                <extra></extra>
                                <pass>2</pass>
                        </encodingParams>
                </task>
                <task>
                        <name>SS 16:9 352p 1100 MP30</name>
                        <qualityKey>main 1300k std afa 24FPS</qualityKey>
                        <resultFilename>video2.mp4</resultFilename>
                        <encodingParams>
                                <videoBitrate></videoBitrate>
                                <audioBitrate></audioBitrate>
                                <gop></gop>
                                <extra></extra>
                        </encodingParams>
                </task>
        </tasks>
    </transcoding>
    <packaging>
        <saveTranscoding>true</saveTranscoding>
        <packages>
              <package>
                        <name>Widevine Trickplay SD</name>
                        <type>WIDEVINE</type>
                        <resultFilename>WV resultVideo</resultFilename>
                        <trickPlayGop>2100</trickPlayGop>
                        <filesGop>2100</filesGop>
```

```
                        <files>
                              <file>
                    <filename>1000 sintel main 480p tp.mp4</filename>
                      <isTrickPlay>Y</isTrickPlay>
                </file>
                <file>
                      <filename>1000 sintel main 480p.mp4</filename>
                      <isTrickPlay>N</isTrickPlay>
                </file>
                <file>
                      <filename>700_sintel_main_480p.mp4</filename>
                      <isTrickPlay>N</isTrickPlay>
                </file>
                       </files>
                </package>
                <package>
                       <name>Video AFA</name>
                       <type>AFA</type>
                       <resultFilename>AFA resultVideo</resultFilename>
                       <trickPlayGop>2100</trickPlayGop>
                       <filesGop>2100</filesGop>
                       <files>
                              <file>
                    <filename>video1.mp4</filename>
                      <isTrickPlay>N</isTrickPlay>
                </file>
                <file>
                    <filename>video2.mp4</filename>
                    <isTrickPlay>N</isTrickPlay>
                </file>
            </files>
                </package>
        </packages>
    </packaging>
    <notifications>
        <start>http://urlWebService/Start</start>
        <done>http://urlWebService/Done</done>
        <error>http://urlWebService/Error</error>
    </notifications>
</job>
```

QualityKey Examples

NicePeopleAtWork's encoding solution has more than 2000 encoding profiles that can be used to encode content to the best quality for a given connected device.

Due to the fact that providing the full list would be unnecessary as well as it will increase this documentation length with no relevant information, only some qualityKeys are provided.

If you need some custom qualities, get in touch with your Customer Engineer to retrieve the best ones:

HD Packaging profiles

SS_16_9_720p_3400_MP32

SS_16_9_720p_2500_MP32

SS_16_9_720p_1700_MP32

SS_16_9_720p_1200_MP32

SD Packaging profiles

SS_16_9_404p_1500_MP30

SS_16_9_404p_1100_MP30

SS_16_9_404p_750_MP30

Nice Packager Widevine response XML

Hereunder you can see an example with the structure of the resulting XML when encrypting a content using Widevine:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xmlstate SYSTEM "/media/externo/PackagerTest/dtd/resultxml.dtd">
<xmlstate>
    <videoname>final 1.wvm</videoname>
    <datemodify>1367568090140</datemodify>
    <videosize>298418340</videosize>
    <assetname>test1</assetname>
    <assetid>809554171</assetid>
</xmlstate>
```

This XML file will be found in the same path than the resulting packaged file. It will have the following name structure: `resultfilename.wvm.xml`, where resultfilename is the name defined in the uploaded XML.

When requesting a license to Nice, the user will have to use the `<assetid>` parameter.

Nonetheless some other interesting parameters are included in that XML which can also be used by the customer application:

- **Videoname:**

  Name of the packaged file

- **Date Modify:**

  Date when the file was created

- **Videosize:**

  Size of the resulting file

- **Asset name:**

  Name given in the uploaded task XML that generated this resulting XML file

Nice Licenses PHP code template

Hereunder, the user can see a PHP template on how to generate the license URL for a given content.

```
<?php
                date_default_timezone_set ('Etc/GMT+0');

                // Variable definition

                // Mandatory variables
                $accountCode = "nicetv";
                $expireTime = 3 * 30 * 7 * 24 * 3600; // In seconds
                $contentId = 810854718;
                $ruleParam1 = "testing1";
                $ruleParam2 = "testing2";

                $serverURL = "http://drm.license.nice264.com/"; // Mandatory variable for
testing purposes with this script, on the final implementation it will stay still with
no modifications

                //Optional variables
                $ip = "";
                $duration = "";
                $purchaseDuration = "";
                $mediaId = "";
                $transactionId = "";
                $countries = "";


                // License Generation
                $licenseForToken =
base64_encode('duration='.$duration.'&purchaseDuration='.$purchaseDuration.'&countries='
.$countries);
                $license = urlencode($licenseForToken);

                // Helper instruction
                // print_r($license);

                // Expire Time Generation
                $expireTime = (time() + $expireTime) . "000"; //miliseconds conversion
                $transactionTime = time();

                // Generación del token
                $token =
md5($ruleParam1.$licenseForToken.$ip.$contentId.$mediaId.$transactionId.$transactionTime
.$accountCode.$expireTime.$ruleParam2);

                // URL definition
                print_r("\n");
                print_r("\n");

                $url =
"$serverURL?license=$license&ip=$ip&contentId=$contentId&mediaId=$mediaId&transactionId=
$transactionId&transactionTime=$transactionTime&accountCode=$accountCode&expireTime=$exp
ireTime&niceToken=$token";

                print r($url);

                print_r("\n");
                print_r("\n");

?>
```