TOA XML COMMUNICATION PROTOCOL

TCP/IP & XML Exchange Protocol for communication with Just Connect (2.0 and later) and just:in (up to v.5.5).

v.2.1 – Effective: October 2023

Disclaimer

ToolsOnAir Broadcast Engineering GmbH ("TOA") provides this document as a general description of the TOA XML Communication Protocol. It is provided to customers and system integrators that have officially licensed (permanently or temporarily), any TOA Product(s) and have a TOA Software Support & Software Maintenance Agreement in effect.

TOA uses reasonable efforts to include accurate, complete and current information in this document, however, TOA does not warrant that the content herein is accurate, complete, current, or free of technical or typographical errors. TOA reserves the right to make changes and updates to any information contained within this document without prior notice.

TOA shall not be responsible for any errors or omissions contained in this document, and in particular TOA shall not be liable for special, indirect, consequential, or incidental damages, or damages for lost profits, loss of revenue, or loss of use, arising out of or related to the information contained in this document, whether such damages arise in contract, negligence, tort, under statute, in equity, at law or otherwise.

Table of Contents

1 Change Log:

Date	Version	Changed by	Changes
10/03/2023	**2.1**	**MGR**	**Updates/changes:** • Updated description to reflect new REST API availability for just:in capture solutions.
01/21/2020	**2.0**	**GD**	**Updates/changes:** • Updated Tags listing • Updated Attributes listing
01/01/2017	**1.0**	**DL**	**Initial Document Release**

2 Definitions and Naming Clarifications:

- "**just:in**", "**just:in multi**", "**just:live**", "**just:play**" all refer to the macOS solutions offered by TOA. If the application is discussed as such (e.g., **Just Connect**, **Just Out**, **Just In Engine**), it is written in the same form as it is displayed in the macOS Finder.
- As of version 5.0 "**just:in multi**" has been renamed to "**just:in mac**" for better clarity, especially when comparing it to "**just:in linux**", running on a TOA appliance using Linux OS as the backend operating system.
- "**Third-party application**" refers to any external (third-party) application that needs to interact/integrate with TOA Solutions.
- As of version 5.5 all **just:in** solutions can directly be accessed via the **TOA REST API**. Descriptions relating to the former **TOA XML Communication Protocol** will be flagged as "**DEPRECATED**".
- The entry-level solutions, namely "**just:in mac lite**" and "**just:in mac lite NDI**" were introduced in Fall 2023 and all feature the **TOA REST API**.

3 Overview:

3.1 Document and Protocol Scope:

Please note that starting with version 5.5, all just:in solutions feature the new TOA REST API allowing developers or system integrators to directly access just:in mac, just:in mac lite, just:in mac lite NDI and just:in linux.

This document is intended for third-party developers or system integrators who may wish to integrate with TOA Solutions and details the TCP/IP & XML protocol used by **Just Connect** and **Just In Engine**, up to version 5.5, to exchange information.

For example, in **Just Connect's** case, a third-party application can send and receive information about the schedule that is then sent on to **Just Out** for playout.

For **Just In Engine**, up to version 5.5, the third-party application can control crash (immediate) recording, select presets and send a batch list for recording, and receive video and audio previews that can be displayed in a user interface if required.

For **just:in** there is no equivalent of **Just Connect**, so the third-party application does communicate directly with **Just In Engine** (**DEPRECATED INTEGRATION**). Starting with version 5.5, all **just:in** capture solutions will utilize the **TOA REST API** for integration and remote-control purposes.

Note that this format is not that used directly by **Just Out** and thus does not enable developers to integrate directly with **Just Out**, bypassing **Just Connect**. Direct communication with **Just Out** is not currently supported by ToolsOnAir. As such, any third-party developer integrating with **Just Out** will also require **Just Connect** running on the network.

This document is intended for developers and system integrators familiar with the concepts of TCP/IP and XML, and with **just:play**, **just:live**, **Just Connect** and/or **Just In Engine** (up to version 5.5) and **just:in multi** (DEPRECATED). It is recommended that developers first install those applications relevant to the proposed integration, read the relevant user manuals and familiarize themselves with the concepts and features of the TOA Solutions before reading this document.

4 The TOA Applications for macOS:

There are several TOA applications for macOS, which can be broadly split into two classes of application:

- **TOA Engine Applications for macOS:** are generally faceless applications (although they do have minimal interfaces containing status and diagnostic information) and do the heavy lifting of playing out and capturing video and audio. The TOA Engines for macOS also natively connect to SDI video devices, or IP-based streams.
- **TOA User Interface(s) for macOS:** are the "windows" onto the engines, allowing the user to control the engine or engines running on the network. The communication between the engines and the user interfaces is network-based, meaning that the engine and the user interface must not be running on the same system. In line with a classic Client-Server model, it is recommended to run the TOA Engines and TOA User Interfaces on separate machines.
- **PLEASE NOTE:** Starting with versions 3.x of both TOA Capture and Playout solutions, there's one common control application, namely Just Control, which unites all former dedicated control applications.

4.1 "TOA Engine Applications for macOS

- **Just Out** is the engine responsible for compositing video and graphics and sending the resulting frames to either a video output card (e.g., AJA or Blackmagic Design) that will then output an SDI signal or an NDI®, SRT, UDP, ST-2110 IP stream signal.
- **Just In Engine** is the engine application for macOS responsible for capturing one or more SDI, NDI®, SRT, UDP, ST-2110 signals originating from a video device card (e.g., AJA, Blackmagic Design) or IP stream and writing these signals into a movie container (currently MOV, MXF or MP4). The frames can be written uncompressed in either 8-bit or 10-bit, or compressed by a specific codec (e.g., Apple ProRes, XDCAM HD 422, AVC-I, H-264, H-265). A single **Just In Engine** application can, depending on the hardware available on the system, support more than one Channel, where a Channel is a single SDI, NDI®, SRT, UDP, ST-2110 signal that can be written to a file. The engine supports **Crash** (including Gang and Split), **Batch** and **Schedule** recording modes as well as VTR (Sony 9-pin) control. All TOA capture solutions based on **Just In Engine** technology (from version 5.5 onwards) can be accessed via the **TOA REST API**.

4.2 User Interface Application for macOS (up to version 2.x):

- **Just Connect** is the Channel manager who also acts as a gateway between **Just Out** and any third-party applications transmitting schedule information. The third-party application sends requests to Just **Connect** and receives acknowledgments back depending on whether the request was successful. The basic unit of organization in **Just Connect** is the Channel, and a single instance of **Just Connect** can manage several Channels. As such, a third-party application does not simply establish contact with **Just Connect,** but rather a specific Channel managed by **Just Connect**. In turn, **Just Connect** will establish connections with the **Just Out** engine or engines assigned to the Channel and forward relevant schedule information to the engines. Furthermore, any communication received Just Out that is relevant to the third-party application will be forwarded on to the application. The third-party application will never itself communicate with **Just Out**. Indeed, such direct communication would be complex for a third-party application to manage because each track in a Channel may, in fact, be rendered by a different instance of **Just Out**, and also by a redundant **Just Out;** thus a single Channel may have ten separate **Just Out** engines running, and a third-party application would have to establish connections with all of them and ensure that each only received the relevant schedule information. By communicating just with **Just Connect**, all this complexity is removed from the third-party application.
- **just:play** is the dedicated user interface intended for scheduled 24/7 Master Control playout. It communicates with **Just Connect** to retrieve schedule information, displays that information for the user and sends requests to **Just** Connect based on the user's actions (e.g., create a new playlist, insert or delete an item in a playlist). As all communication is network-based, it is not necessary for **just:play** and **Just Connect** to be running on the same system, they must simply be on the same network. Additionally, several instances of **just:play** may be viewing the same Channel at the same time. Any changes made by one **just:play** will be instantly reflected on all other instances.
- **just:live** is the user interface intended for Live Production playout where **Just Out** reacts as quickly as possible to user commands - generally within a few frames. Apart from the reactivity, the interface itself is organized differently to account for the different demands placed on an operator in a Live Production situation rather than a pre-defined, scheduled operation. Otherwise, all the features of **just:play** described above hold true for **just:live**.
- **just:in multi** is the interface intended for multichannel capture. It can control multiple Channels coming from multiple instances of **Just In Engine** on the network (e.g., two Channels from one instance of **Just In Engine,** two from another and a fifth Channel from a 3rd instance of **Just In Engine**). The user interface provides live feedback about the status of the Channels (video and audio previews) and diagnostics about the systems on which the **Just In Engine** are running (available memory, space on the storage systems and so on). The user can then start and stop recording, build and execute batch lists, control any connected videotape machines (VTR control) and view any pre-defined schedules due to be recorded.

4.3 User Interface Application for macOS (from version 3.x onwards):

- Starting with versions 3.x of both TOA Capture and Playout solutions, there's one common control application, namely **Just Control**, which unites all former dedicated control applications listed above in section 4.2.

5 Integrations strategies

A third-party application integrating with TOA Solutions can fall into one of two broad categories:

- As a complete replacement for one or more of the TOA User Interfaces (**just:play**, **just:live** and **just:in multi** – all within **Just Control**). In this scenario, it is likely that the third-party application will itself provide a user interface and a means for the user to control the playout (using the **TOA XML Protocol**) or capture (using the **TOA REST API**). Only **Just Connect** and **Just Out** (for playout) and/or **Just In Engine** (for capture) will be actively running on the network.
- As a faceless backend solution providing schedule information that the user will then view and potentially edit in **just:play** or **just:live** interfaces within **Just Control** in addition to controlling the playout.
- Note that this strategy is only available for integration with **Just Connect** (Playout).

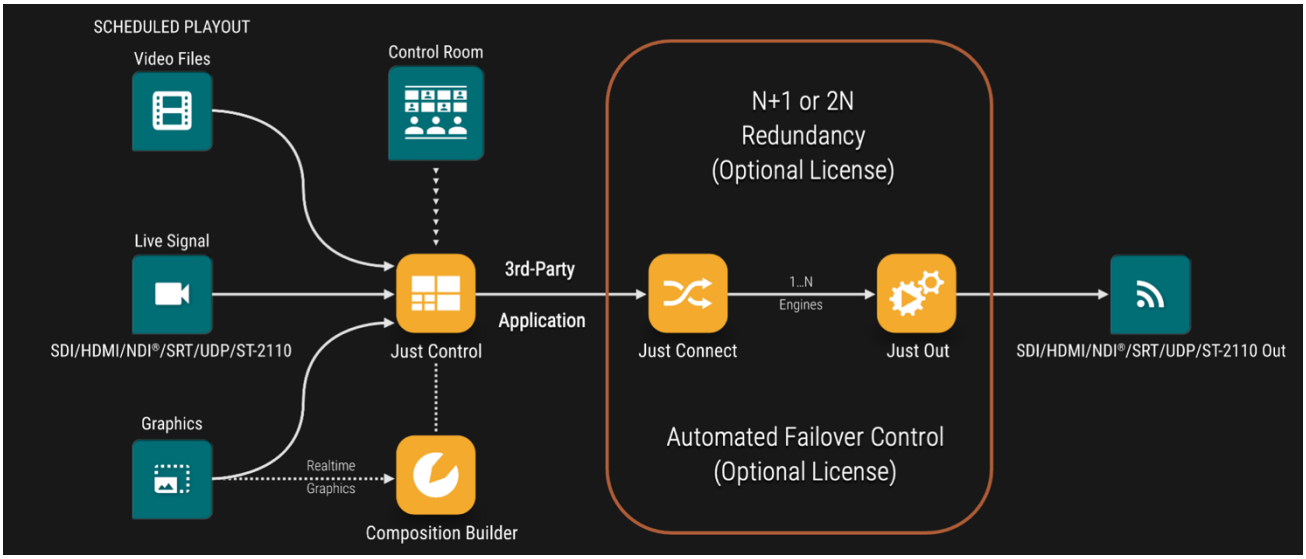
(DEPRECATED): Due to the architecture of **just:in**, it is not possible to send information from a third-party application to **just:in** that can then be viewed with the **just:in multi** user interface.

Any third-party application has the same "status" in the TOA Solutions ecosystem as the **just:play**, **just:live** or **just:in multi** interfaces (all within **Just Control**). As outlined above, with **just:play** and **just:live** it is perfectly possible to have both the third-party application and just:**play / just:live** running at the same time, connected to the same Channel via **Just Connect**.

And just as with two instances of **just:play or just:live** editing the same channel, any third-party application will receive full notification of any user actions in **just:play or just:live,** allowing it to stay fully synchronized. However, there are some additional requirements placed on a third-party application in terms of the XML that it transmits if **just:play or just:live** are being used to view the schedule. Please refer to the sections on the "Schedule XML Format" for details of these requirements.

In either case, it is highly recommended that a third-party developer should first install the related TOA Solution he wants to integrate with (just:play, just:live – using the TOA XML Protocol and/or just:in mac, lite or linux – using the TOA REST API) to familiarize themselves with the concepts and solutions described above.

Furthermore, just:play can be a very useful tool to produce example schedule XML. For instance, if you want to see the schedule XML for a specific playlist, you can build the playlist in the just:play interface and then export the playlist as a file which will then contain exactly the XML that Just Connect would expect from a third-party application wishing to create such a playlist. Please refer to the just:play user guide for full details on creating and exporting playlists.



6 Communication with TOA Solutions over TOA XML Protocol

Once the **Just Connect** or **Just In Engine** (up to version 5.5) application is started, it will automatically broadcast a TCP service using Bonjour. External applications can use Bonjour services to discover the IP address and port used by **Just Connect** or **Just In Engine** (up to version 5.5), as follows:

Application	Bonjour Service	Bonjour Name
Just Connect	_toaautomator._tcp.	The name of the channel as defined in **Just Connect** (e.g., "My Channel")
Just In Engine*	_toajustin._tcp.	The system host name of the machine on which the **Just In Engine** is running (e.g., "My-Mac-Pro")

*= DEPRECATED as of version 5.5 of all **just:in Capture Solutions**. Please refer to the **TOA REST API** for integration tasks.

There will also be a domain associated with the Bonjour service. In most cases, this will simply be "**. local**" but may be different depending on the specific network and DNS configuration.

For example, a third-party application wishing to connect to a specific Just Connect Channel should filter for Bonjour services with the type "**_toaautomator._tcp.**" and then check the service's name against the wished-for Channel name.

As **Just Connect** and **Just In Engine** both support more than one Channel simultaneously, it is possible that one instance of **Just Connect** or **Just In Engine** will broadcast several TCP services on the same IP address but with unique ports for each individual channel. Currently, it is not possible to assign a Channel a fixed port number, so Bonjour is the only way to discover a Channel's port. This may be addressed in a future release to allow integration with systems that do not support Bonjour directly, such as Windows or Linux (although Bonjour services may also be available on these systems with additional software installed).

All communication between **Just Connect** or **Just In Engine** and third-party applications use plain UTF-8 encoded text. A zero byte is sent to indicate the end of a message. Most communication involves sending an XML document with a zero terminator at the end; however, some basic commands are sent as simple text strings.

When connecting to **Just Connect** or **Just In Engine,** an application will immediately receive the string "handshake" with a zero byte terminator. The application should immediately send back the same "handshake" string to complete the connection. Then the application should send three null-terminated strings to identify the application, as follows:

Application	Direction	Data
Just Connect / Just In Engine	Sends	"handshake\0"
Application	Receives	"handshake\0"
Application	Sends	"handshake\0"

6.1 Application identification in Just Connect

In addition to the handshake detailed above, applications connecting to **Just Connect** must further identify themselves before the stream is fully available. To identify itself, the application must send the following three null-terminated strings:

Application	Direction	Data
Application	Sends	"host xxx\0"
Application	Sends	"ipv4 a.b.c.d\0"
Application	Sends	"appID yyy\0"

Please note:

- The "**xxx**" string following "**host**" should be the host name of the system running the application. For macOS systems this should be the "**Computer Name**" defined in the "**Sharing**" System Preferences/Settings panel.
- The "**ipv4**" string should be followed by the standard format IPv4 address (e.g., 192.168.0.10) of the system running the application.
- The "**yyy**" string following "**appID**" should identify the application. This should normally be "**just:play**" for applications controlling a 24/7 Master Control Channel or "**just:live**" for applications controlling a Live Production Channel

6.2 User authorization in Just Connect

In addition to the handshake, **Just Connect** also requires a basic form of user authorization. Once the handshake is complete, the third-party application should send a string with the format "**user xxx**" where "**xxx**" is the name of the user wishing to connect to **Just Connect**. The user name must be one of the names defined in the channel's list of users in **Just Connect**. If the user is recognized by **Just Connect,** it will send back the string "**authorized**", after which **Just Connect** is ready to receive commands from the application. If the username isn't recognized, **Just Connect** will send the string "**notAuthorized**" and any further communication from the application will be ignored by **Just Connect**.

Assuming that the user is recognized, the sequence would be as follows:

Application	Direction	Data
Application	Sends	"user user_name\0"
Just Connect	Receives	"user user_name\0"
Just Connect	Sends	"authorized\0"
Application	Receives	"authorized\0"

If the user has a password set in **Just Connect**, then **Just Connect** will respond with "**passwordRequired**". In this case, the client must respond with the SHA1 digest of the user's password. If the password matches, then **Just Connect** will respond with "**authorized**", or if the password is incorrect with "**notAuthorized**". The digest should be formatted as 20 hexadecimal bytes, as shown in the following code snippet:

```
- (NSString *)sha1Digest
{
	unsigned int i;
	unsigned char digest[SHA_DIGEST_LENGTH];
	const char *string = [self UTF8String];
	SHA1((const unsigned char *)string, strlen(string), digest);
	NSMutableArray *array = [NSMutableArray array];
	for (i = 0; i < SHA_DIGEST_LENGTH; i++) {
		[array addObject:[NSString stringWithFormat:@"%02x", digest[i]]];
	}
	return [array componentsJoinedByString:@""];
}
```

The full sequence in this case would be:

Application	Direction	Data
Application	Sends	“user user_name\0”
Just Connect	Receives	“user user_name\0”
Just Connect	Sends	“passwordRequired"
Application	Receives	“passwordRequired”
Application	Sends	“password xxxxx” (SHA1 digest)
Just Connect	Receives	"password xxxxx"
Just Connect	Sends	“authorized\0”
Application	Receives	“authorized\0”

As TCP/IP communication is socket-based, terminating a session with Just Connect is as simple as closing the underlying socket.

6.3 Channel locking in Just In Engine (DEPRECATED)

Contrary to **Just Connect**, **Just In Engine** does not have any user authorization. However, before using a Channel, the client must first lock it. A client can discover Channels by sending the “**requestChannel**” message to **Just In Engine**.
From the list of available Channels, the client can then request locks on one or more channels by sending the “**requestLock**” message.
Please refer to the “**Ingest communication protocol**” section (DEPRECATED) or to the TOA REST API documentation for a full description of the available messages.

6.4 Heartbeat Messages

Both **Just Connect** and **Just In Engine (up to version 5.5)** will send heartbeat messages to all connected client applications (including the third-party application). A heartbeat message is a plaintext message with the prefix “**heartbeat**”, then a space character and then the current timecode in standard SMPTE timecode format (“**00:00:00:00**”). The client application can use this timecode to synchronize its internal time with the engine’s time and optionally display this information to the user (the current timecode and/or the mere fact that the engine is sending heartbeats to the client).

7 Communication with TOA Capture Solutions over TOA REST API

Starting with version 5.5 of the **just:in Capture Solutions** for either macOS or Linux, all third-party system integrations are achieved by using the **TOA REST API**.

The new **TOA REST API** provides almost all the features of the well-known **Just Control** user interface. The **TOA REST API** can easily be tested with the Postman.app for macOS or with the integrated **Swagger-UI**.

Please note that the API calls are received by the **Just In Engine** application, therefore the **IP Address** of the machine running the **Just In Engine** application must be used.

To gather more detailed information about the **TOA REST API**, please visit one of the following links:

TOA REST API for just:in mac Solutions:
https://toolsonair.atlassian.net/wiki/spaces/TST/pages/3940320179/JIM+ToolsOnAir+REST+API+v.6.5

TOA REST API for just:in linux Solutions:
https://toolsonair.atlassian.net/wiki/spaces/TST/pages/3662020862/JIL+ToolsOnAir+REST+API+v.6.1

8 Playout schedule XML format

The schedule held by **Just Connect** is defined by a tree-structure of nodes, with different "classes" of **<node>** used to represent logical structure in the schedule. In addition to a class, or type, each **<node>** has several attributes or metadata. For each class of **<node>** there are some mandatory attributes and some optional attributes.

In the XML description, the schedule is represented by three tags:

- The **<node>** tag is the fundamental tag in the structure and defines a single **<node>**. A **<node>** can represent a day, a playlist, a track, an item to play out or a trigger to a stop frame on a **Composition Builder** or Quartz Composer graphic file.
- The **<attribute>** tag defines one attribute (or property or metadata) on a **<node>**.
- The **<resource>** tag defines a **<node>**'s resource, or media, such as a video file or a **Composition Builder** or Quartz Composer graphic file.

Child nodes are represented by **<node>** tags nested in the parent's **<node>** tag. Each **<node>** tag can have any number of child **<attribute>** tags. This gives the following structure:

```
<node>
	<attribute>...</attribute>
	<attribute>...</attribute>
	<resource>...</resource>
	<node>
		<attribute>...</attribute>
		<attribute>...</attribute>
		...
	</node>
</node>
```

Each **<node>** tag contains nested **<attribute>** tags to define properties, **<node>** tags to define child nodes and, optionally, one **<resource>** tag to define the **<node>**'s resource (video file, **Composition Builder** or Quartz Composer file).

8.1 The <node> Tag

The **<node>** tag is used to define a **<node>** in the tree and has three required attributes:
The "**id**" attribute is a string that defines a unique identifier for the **<node>**. Every **<node>** defined in the schedule must have an entirely unique **ID** identifying it. It is the responsibility of the third-party application communicating with **Just Connect** to assign and manage **<node> IDs**.
The "**class**" attribute is an integer that defines the class, or type of **<node>** being defined. The following values are defined:

Class	Name	Definition
0	Project	Reserved for internal use, do not use.
1	Day	Defines all the playlists for a single day in the schedule.
2	Playlist	Defines a logical block of items to play.
3	Graphic Track	A graphic track can only contain play nodes with **Composition Builder** or Quartz Composer files. Several graphic tracks may be defined for a given Channel, and they are rendered in the order specified in **Just Connect**.
4	Video Track	A video track can only contain play nodes with video files. Currently, **Just Connect** and **Just Out** only support a single video track per channel. This may be changed in future versions.
5	Play	Defines a single item to play in a playlist. Depending on the parent track type can define a **Composition Builder** file, Quartz Composer file or video file as its resource.

6	Trigger	Defines a "**stop frame**" trigger for a **Composition Builder** or Quartz Composer graphic. As such, this type of **<node>** should only be a child for a play **<node>** with a graphic resource.
7	Real-time Playlist / Folder	A special type of playlist or folder used to group nodes for "live" situations. See the section on "Live Playout" for more details.
8	Real-time Play	A special type of play **<node>** used for "live" situations. See the section on "Live Playout" for more details.

The "flags" attribute is a bit mask of the following values:

Value	Definition
0x00000001	A **<node>** is chained (starts immediately after the previous **<node>** ends). Is relevant for playlist and play nodes.
0x00000008	Immediate play. The **<node>** should be played as soon as possible (the earliest frame available). Is relevant in "live" situations. See the section on "Live Playout" for more details.
0x00000010	Immediate cue. The **<node>** should be cued (the first frame of the **<node>**, without playing any further frames) as soon as possible (the earliest frame available). Is relevant in "live" situations. See the section on "Live Playout" for more details.

Thus, a chained container **<node>** could be defined as follows:

```
<node id="1234" class="2" flags="1">...</node>
```

Note that if the value of "flags" is "zero" this attribute may be omitted in the XML as the default value of flags is "zero".

In addition to the three required attributes described above, all "**<track>**" nodes (with class="**3**" or class="**4**") have a further required attribute:

- The "**trackId**" attribute is a string that defines both the content of the **<track>** and its position in the render sequence. The format is a letter followed by a number. The letter "v" defines a video **<track>** while "g" defines a graphic **<track>**. The number is a "zero"-based index, whereby tracks with lower values are rendered first, putting them at the "back" of the composited frame.

Under normal circumstances the following tracks are defined (in render order): **"v0", "g0", "g1", "g2", "g3"**.
This results in a single video **<track>** in the background with four (4) graphic tracks rendered on top.

Please note that the order in which the **<track>** nodes are defined in the schedule XML is irrelevant as **Just Connect** uses the "**trackId**" attribute to send the **<track>**'s contents to the **Just Out** engine assigned to the **<track>** and the render order is defined by the Channel itself, not the order that the tracks appear in the schedule.

Thus, a track **<node>** containing video play nodes to be rendered on the video track "**v0**" would be defined as:

```
<node id="99" class="4" trackId="v0">
```

The class attribute of "**4**" defines a video **<track>**, and the "**trackId**" of "**v0**" defines the actual logical video **<track>**.

Nodes of class "**8**" (real-time play) also have two additional attributes:

- The "**reference**" attribute defines the ID of the play **<node>** that should be played out. This play **<node>** must have already been loaded into the real-time "**workbench**" or "**contents**" playlist.
- The "**trackId**" defines the **<track>** on which the play **<node>** should be played out. Standard values are "**v0**" for the video **<track>** and **"g0", "g1", "g2" and "g3"** for the graphic tracks.

For example, to schedule the play **<node>** with the ID "**RESOURCE**" on the track "**v0**" the **<node>** would be defined as:

```
<node id="REALTIME_PLAY" class="8" reference="RESOURCE" trackId="v0">
```

For full details on how play nodes are loaded, referenced and played out in a live situation, please refer to the section on "Live Playout" later in this document.

8.2 The <attribute> Tag

Not to be confused with XML tag attributes, the **<attribute>** tag defines an attribute, or property, for a **<node>**. As with the **<node>** tag, the **<attribute>** tag has three required attributes in the following format:

```
<attribute key="xxx" type="y" flags="z">value</attribute>
```

The "key" attribute is a string that defines which of the **<node>**'s properties is being defined. The values are described in detail later in this section.

The "type" attribute is an integer that defines what kind of property is being defined, one of the following values:

Value	Name	Definition
0	String	A single UTF-8 encoded string.
1	Integer	A 32-bit integer value.
2	Double	A double (floating point) value.
3	Bool	A true / false Boolean value.
4	Index	An integer value from a list of possible values.
5	Dictionary	Reserved for future use, do not use.
6	Array	An array of double or timecode values.
7	Timecode	A timecode value, with or without date.
8	Color	An RGBA color value.

The individual attribute types are described in full detail later in this section.

The "flags" attribute is an integer bit mask containing any of the following values:

Value	Name	Definition
0x00000001	Read only	The attribute will be displayed in **just:play** or **just:live** interfaces but disabled so that the user cannot alter the value.
0x00000002	Input port	Set to indicate that the attribute represents an input port value for a **Composition Builder** or Quartz Composer graphic file. In this case, **the attribute's key must match exactly the input port's key as published in Composition Builder or Quartz Composer and the attribute's type the input port's expected data type.**
0x00000004	Track Control	Set to indicate that the attribute represents an infinite graphic **<track>** control setting (whether an infinite graphic **<track>** is enabled or disabled while the **<node>** is playing out). In this case, the attribute's key should have the prefix "**toaInfiniteTrack**" with the **<track> ID** appended (e.g., "**g0**", "**g1**" etc.) For example, if the graphic **<track>** with the **ID** "**g0**" is defined in **Just Connect** as being an infinite graphic **<track>**, then an attribute with the key "**toaInfiniteTrackg0**" would specify whether the graphic on this **<track>** would be displayed or not. The attribute must be a Boolean attribute with "**yes**" or "**true**" specifying that the infinite graphic will be disabled and "**no**" or "**false**" that the infinite graphic will be enabled (this is also the default value if the attribute is omitted). This type of attribute is valid for class "**2**" (Playlist) and class "**5**" (Play) nodes.
0x00000008	Hidden	Set to indicate that the attribute should not be displayed in the **just:play** or **just:live** interfaces for editing by the operator.
0x00000010	Custom Interface Input Port	Used in combination with the "**Input Port**" flag to indicate that the input port in question is a port controlled by a "**custom interface**" within **just:live** (an HTML-based interface displayed in a separate window). Such attributes are not displayed in the **just:live Inspector**. Instead, a "**Custom Interface**" group is added to the **Inspector**, with a button that displays the custom interface when clicked by the user. Please refer to the **just:live** user guide for further details on custom interfaces.
0x00000020	Metadata	Set to indicate that the attribute is metadata for the **<node>**. This is simply a hint for the **just:play** and **just:live** interfaces to display all metadata attributes together in a separate group.

In addition, the flag bits **"12-15" (0x00001000 to 0x0000F000)** can be used to specify logical ordering of attributes when they are displayed in the **just:play** or **just:live** interfaces. Attributes are sorted numerically according to the values specified in these order bits, and then attributes with the same order bits are sorted alphabetically by name.

Note that if the value of "flags" is "**zero**" this attribute may be omitted in the XML as the default value of flags is "**zero**".

In addition to the required attributes, the tag also has two optional attributes:

- The "**name**" attribute defines a human-readable label for the attribute (instead of the "**key**" attribute" which may not be meaningful to the user).
- The "**category**" attribute defines a category, or group, of attributes to which the attribute belongs.

Both these attributes are only relevant if the schedule will be presented to a user in the **just:play** or **just:live** interfaces, and if the attribute in question is a custom attribute. The "**name**" attribute will be used when displaying the attribute's value to the user, and attributes of the same category will be grouped together in the user interface (a "**group**" in the **Inspector**). A third-party application sending custom (non-standard) attribute keys, where the user is expected to view the schedule in the **just:play** or **just:live** interfaces, should always transmit names and categories for such attributes. It is never required to send names or categories for any of the standard attribute keys described in this document, as they are pre-defined within the **just:play** and **just:live** interfaces. Equally, if the third-party application has its own user interface and the schedules transmitted to **Just Connect** will not be viewed or edited with **just:play** or **just:live,** then the name and categories can be safely omitted.

For example, a string attribute used to set an input port on a Quartz Composer graphic with the port key "**myInputKey**" would be defined as:

```
<attribute key="myInputKey" type="0" flags="2" name="Title" category="Input Port">Hello World</attribute>
```

Here the attribute's key matches the input port's key, the type is "**0**" for a string attribute and flags is "**2**" to indicate an input port attribute. If displayed in the **just:play** or **just:live** interfaces, the attribute will be in the "**Input Port**" **Inspector** group with the label "**Title**" and a text entry field with the initial value "**Hello World**".

8.2.1 String Attributes

The tag's text defines the attribute's string value, as in "**Hello World**" in the following example:

```
<attribute key="xxx" type="0">Hello World</attribute>
```

8.2.2 Integer Attributes

The tag's text defines the attribute's integer value, as in the value "**1000**" in the following example:

```
<attribute key="xxx" type="1">1000</attribute>
```

8.2.3 Double Attributes

There are two possible formats for a double attribute. In the basic format, the tag's text simply defines the attribute's double value, as in the value "**3.14**" in the following example:

```
<attribute key="xxx" type="2">3.14</attribute>
```

However, it is also possible to define a minimum and/or maximum value for the attribute by defining **<min>** and/or **<max>** child tags. In this case, the value itself is defined as the text of a **<value>** tag.

For example, to define the attribute with a value of "**3.14**" and a maximum value of "**100.0**" use the following:

```
<attribute key="xxx" type="2"><max>100.0</max><value>3.14</value></attribute>
```

To additionally define a minimum value of "**0.0**", use the following:

```
<attribute key="xxx" type="2"><min>0.0</min><max>100.0</max><value>3.14</value></attribute>
```

Note that the minimum and maximum values are optional and are only relevant if the schedule is to be presented to the user for further editing. In this case, when the attribute is displayed in the **Inspector**, the user's entry can be validated against the given range of values to ensure that invalid values are never sent to **Just Out**.

8.2.4 Boolean Attributes

The tag's text defines the attribute's Boolean value with "**F**" representing "**false**" or "**no**" and "**T**" representing "**true**" or "**yes**". The following example defines a "**true**" attribute:

```
<attribute key="xxx" type="3">T</attribute>
```

8.2.5 Index Attributes

Index attributes define an integer value from a range of values, with optional names defined for each value that may be presented to the user in the form of a list to select from. There are two possible formats for this attribute. In the basic form, the tag's text simply defines the attribute's integer value, as in the value "**1**" in the following example:

```
<attribute key="xxx" type="4">1</attribute>
```

To define a list of values, the following tags are defined:

```
<values>
      <string>Name for value 0</string>
      <string>Name for value 1</string>
      <string>Name for value 2</string>
</values>
<max>2</max>
<value>1</value>
```

Where the **<string>** tags define a list of names for the values (note that index attributes are always "zero"-based), the **<max>** tag's text defines the maximum value allowed and the **<value>** tag's text defines the currently selected value for the attribute.

The following example defines an index attribute with the names "**Zero**", "**One**" and "**Two**" where "**One**" is currently selected:

```
<attribute key="xxx" type="4">
      <values>
            <string>Zero</string>
            <string>One</string>
            <string>Two</string>
      </values>
      <max>2</max>
      <value>1</value>
</attribute>
```

Please note that defining value names for an index attribute is optional and only relevant if the schedule is to be presented to the user for further editing in the **just:play** or **just:live** interfaces. In this case, when the attribute is displayed in the **Inspector**, a drop-down list is displayed from which the user can select the required value. Such a list with strings is obviously easier for the user than having to enter the correct number.

8.2.6 Dictionary Attributes

Not supported in the version 2.x builds of TOA Playout Solutions.

Please do not use and contact TOA for further information.

8.2.7 Array Attributes

The TOA Playout Solutions (**just:live or just:play**) allows arrays of either double or timecode values. The following example defines an array of "**5**" double values with the values **[0.0, 1.0, 2.0, 3.0 and 4.0]**:

```
<attribute key="xxx" type="6">
      <array>
            <double>0.0</double>
            <double>1.0</double>
            <double>2.0</double>
            <double>3.0</double>
            <double>4.0</double>
      </array>
</attribute>
```

The following example defines an array of timecode values with the values **[00:10:00:00, 00:20:00:00 and 00:30:00:00]**

```
<attribute key="xxx" type="6">
      <array>
            <timecode>00:10:00:00</timecode>
            <timecode>00:20:00:00</timecode>
            <timecode>00:30:00:00</timecode>
      </array>
</attribute>
```

8.2.8 Timecode Attributes

The text of this attribute defines a time code and, optionally, a date. The basic time code only format uses the standard SMPTE "**hh:mm:ss:ff**" (hours, minutes, seconds, and frames) format, as in the following example that defines a time code of "**22:30**", or **"10:30 PM"**:

```
<attribute key="xxx" type="7">22:30:00:00</attribute>
```

When a date is also included, it must be in the format "**dd.mm.YYY**" without any spaces and should precede the time code followed by a space, for example:

```
<attribute key="xxx" type="7">1.4.2009 22:30:00:00</attribute>
```

Additionally, it is valid to specify the timecode part as a simple integer, which in this case is the number of frames elapsed since midnight. For example, at "**25 fps**" there are **"90000 frames per hour (25 * 60 * 60)**", so to specify a timecode of "**02:00 AM**" the correct value would be **"180000 (90000 * 2 hours elapsed since midnight)"**. For example:

```
<attribute key="xxx" type="7">1.4.2009 180000</attribute>
```

Note that time codes are always assumed to be in the time zone "**local**" to the Channel itself (i.e., the system time in which **Just Out** is running) and in the correct frame rate for the Channel (i.e., "**25 fps**", "**29.97 fps**", "**30 fps"** etc.). Time codes are not further verified by **Just Connect** or **Just Out**. It is the responsibility of the sending application to verify that the time codes are correct.

8.2.9 Color Attributes

The tag's text defines a color with RGBA components in one of two possible formats. The first format is in hexadecimal with a leading "**#**" character ("web" style). The second format defines the components as floating-point values between "**0.0**" and "**1.0**" separated by commas. In both cases, the components are defined in the order "**red**", "**green**", "**blue**", and "**alpha**". For example, both the following would define a "**red**" color with an "**alpha**" value of "**1.0**" (i.e., opaque):

```
<attribute key="xxx" type="8">#ff0000ff</attribute>
<attribute key="xxx" type="8">1.0,0.0,0.0,1.0</attribute>
```

8.3 The <resource> Tag

All class "**5**" (Play) nodes must define a single **<resource>** tag to define the type of resource or media to be played out by the **<node>**.

The tag has a single "**type**" attribute to indicate the type of resource, with the following values:

Value	Type	Definition
0	QuickTime Movie	The text of the **<resource>** tag defines the name of a video file.
1	Composition Builder/ Quartz Composer Graphic	The text of the **<resource>** tag defines the name of the Quartz Composer or **Composition Builder** file.
2	Live Input	Defines that the live input signal should be displayed. Note that the output card must be correctly configured, e.g., the downstream keyer activated for an AJA card) and a valid SDI signal connected to the card's input for this function to work correctly. The text of the **<resource>** tag is ignored in this case.
3	Image	A special form of Quartz Composer graphic, a still image "wrapped" in a pre-defined QTZ file. The text of the **<resource>** tag defines the name of the still image ("JPEG", "TIFF" or "PNG" format).
4	Graphic Movie	A special form of Quartz Composer graphic, a QuickTime movie file "wrapped" in a pre-defined QTZ file and rendered on a graphic **<track>** instead of the video **<track>**. The text of the **<resource>** tag defines the name of the video file.
5	Gap	Intended to intentionally insert gaps into a video **<track>** if this is desired. Normally, all video play nodes must be chained, and this excludes the possibility of gaps between movie files. Inserting a play **<node>** with a "gap" resource allows such a gap to be inserted between two play nodes with movie resources. The text of the **<resource>** tag is ignored in this case.
6	JavaScript Event	An event that when triggered will compile and execute the JavaScript fragment defined by the "**toaEventScript**" key on the node (**DEPRECATED**).
7	Graphic based on TOA graphic template	The text of the **<resource>** tag defines the name of the **"TOA Graphic"** template.
8	Graphic Movie based on TOA graphic template	A special form of a "**TOA Graphic**", a QuickTime movie file "wrapped" in a pre-defined TOA graphic file and rendered on a graphic **<track>** instead of the video **<track>**. The text of the **<resource>** tag defines the name of the video file.
9	Placeholder	A special "**placeholder**" **<node>** used to mark a place within a playlist that will be filled by a movie later. If the placeholder is not replaced by a movie (resource type "**0**") before the node's start time, then nothing will be played out.
10	Audio	The text of the **<resource>** tag defines the name of the audio file.
11	Workflow	The text of the **<resource>** tag defines the name of onCore workflow project file **(DEPRECATED).**

For example, to define a QuickTime resource with the filename "**Hello.mov**":

```
<resource type="0">Hello.mov</resource>
```

To define a **Composition Builder** resource with the filename "**World.composition**":

```
<resource type="1">World.composition</resource>
```

To switch to live input (e.g., the live SDI input attached to the AJA or Blackmagic Design card's input):

```
<resource type="2"/>
```

8.3.1 Relative and Absolute Resource Files and Repository Folders

Just Out must be configured with one or more video repository folders and one or more graphic repository folders. As such, any files specified by the **<resource>** tags can be relative to these folders. For example, the tag…

```
<resource type="0">Hello.mov</resource>
```

…will cause **Just Out** to search each of the video repository folders until it finds a file with the name "**Hello.mov**". Note that **Just Out** will not search any sub-folders in the repository folders for the file. You can, however, specify folder names relative to any repository folder.

For example, the tag…

```
<resource type="0">Folder/Hello.mov</resource>
```

…will cause **Just Out** to search in all video repository folders for a sub-folder with the name "**Folder**" and, if found, for a file is this sub-folder with the name "**Hello.mov**".

Please note that if the file with the same name exists in multiple repository folders, then **Just Out** searches the repository folders in the order specified in the **Just Out** System Preferences/Settings pane and will take the first file found.

For type "**0**" resources the video repository folder(s) will be searched by **Just Out**, for type "**1**", "**3**" and "**4**" the graphic repository folder(s) will be searched.

It is also valid to specify an absolute filename with the "**/**" character at the start of the filename. For example:

```
<resource type="0">/Volumes/Video/Hello.mov</resource>
```

In this case, **Just Out** will ignore all repository folders and use the file specified.

Please note that if the file does not exist, **Just Out** will not search additionally in its repository folders. Regardless of relative or absolute, it is the responsibility of the 3rd-party application to ensure that all resources transmitted in the schedule are available to **Just Out** at the scheduled playout time.

8.3.2 Attribute Key Names

The **<attribute>** tag's "**key**" attribute defines which of the **<node>**'s properties are defined by the **<attribute>** tag. Some attributes are required, others are optional, depending on the class of **<node>**. All the valid attributes keys are defined below. The table headings for each attribute key have the following meanings:

Attribute Type	The type of data expected for the attribute, one of the types described in the previous sections (**"String", "Integer", "Double", "Boolean", "Array", "Index", "Timecode" or "Color"**).
Definition	Information about the purpose of the attribute.
Required For Node Class	Which classes of **<node>** require this attribute for **Just Connect** and **Just Out** to correctly process the **<node>**.
Optional For Node Class	Which classes of **<node>** may optionally use this attribute, but where the presence of the attribute is not critical for **Just Connect** and **Just Out** (for example, where there is a default setting, or where the attribute is only informational).
Required for UI	Some optional attributes are only then required when the user wishes to view and/or edit the schedule uses TOA's user interfaces (**just:play** or **just:live**). If the third-party application integrating with TOA Solutions also provides a user interface, then it may not be necessary to use the **just:play** or **just:live** interfaces at all. In this case, any attributes marked as optional for a specific **<node>** class but also marked as required for UI can be completely omitted when sending schedules to **Just Connect**. However, if the third-party application is only a faceless backend product with no user interface, and it is intended that the user should use the **just:play** or **just:live** interfaces to view or edit the schedules then all attributes marked as required for UI **must be sent by the application** to **Just Connect**. **Failure to do so will result in unexpected behavior in just:play or just:live, up to and including software crashes**. Note also that optional attributes that are marked as not required for UI will not be available for editing by the user in the **just:play** or **just:live** interfaces if the third-party application does not specify them. Any attributes intended to be edited by the user in the **just:play** or **just:live** interfaces must always be transmitted by the application, even if they are optional and not specifically required for the UI.

8.3.3 "toaName" Attribute

Attribute Type	String
Definition	Defines the name of the **<node>**. The name is informational only (for example, for display in the **just:play** or **just:live** interfaces) and is not used directly by **Just Out**.
Required For Node Class	None
Optional For Node Class	"**2**" (Playlist), "**5**" (Play)
Required for UI	Yes

8.3.4 "toaStart" Attribute

Attribute Type	Timecode **(including date information)**
Definition	Specifies when the **<node>** will start to play out.
Required For Node Class	"**1**" (Day), "**2**" (Playlist), "**5**" (Play), "**6**" (Trigger)
Optional For Node Class	"**5**" (Play)
Required for UI	Yes

Please note that this attribute is specified as required and optional for node class "**5**" (Play). This is due to the difference between scheduled Master Control and Live Production playout. The start timecode is required for all play nodes in scheduled Master Control playout but should not be specified when transmitting a play **<node>** intended for live playout. Please refer to the section on "Live playout" for further details.

8.3.5 "toaDuration" Attribute

Attribute Type	Timecode **(excluding date information)**
Definition	Defines how long the **<node>** will be played out for.
Required For Node Class	"**2**" (Playlist), "**5**" (Play), "**6**" (Trigger)
Optional For Node Class	None
Required for UI	Yes

8.3.6 "toaNaturalDuration" Attribute

Attribute Type	Timecode **(excluding date information)**
Definition	Defines the "**natural duration**" of the **<node>**, or the original duration of the media specified for the **<node>**. For example, for a play **<node>** with a QuickTime movie as its resource, this would be the full length of the media in the movie file, for a **Composition Builder** graphic it is the length of the timeline as specified by the user. This information is required for a user interface where the user can alter the duration of the **<node>** by, for example, adjusting the in- and out-points of a video file. In this case, the duration of the **<node>** specified by the "**toaDuration**" attribute may be shorter than the natural duration of the media. Therefore, any interface editing such a **<node>** must know the natural duration to ensure that the duration doesn't violate the media's natural duration (a movie's duration can never be longer than the natural duration, a **Composition Builder** or Quartz Composition graphic with stop frames can never have a duration shorter than the natural duration).
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	Yes

8.3.7 "toaInPoint" Attribute

Attribute Type	Timecode **(excluding date information)**
Definition	Defines the in-point, or the starting frame, within a QuickTime movie file.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	Yes

Optional for play nodes with QuickTime movie resources, invalid for all other resource types. If not specified, the movie will begin playing from the first frame in the file.

8.3.8 "toaOutPoint" Attribute

Attribute Type	Timecode **(excluding date information)**
Definition	Defines the out-point, or the final frame within a QuickTime movie file, to be played out. The duration is calculated as the out-point minus the in-point.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	Yes

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.
Please note that the **just:play** and **just:live** interfaces incorrectly treat the out-point as "exclusive" - in other words, the out-point specifies the first frame not to be played out instead of the last frame to be played out as in most video editing software.

8.3.9 "toaFadeIn" Attribute

Attribute Type	Double
Definition	Defines how long, in seconds, the QuickTime movie file should fade in (either from black or from transparent over the live input, depending on the value of the "**toaTransparentFade**" attribute). If not specified, the default value is "**zero**", meaning no fade in (instant cut from either the previous movie or live input).
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.10 "toaFadeOut" attribute

Attribute Type	Double
Definition	Defines how long, in seconds, the QuickTime movie file should fade out (either to black or to transparent over the live input, depending on the value of the "**toaTransparentFade**" attribute). If not specified, the default value is "**zero**", meaning no fade out (instant cut to either the next movie or live input).
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.11 “toaAudioFade” Attribute

Attribute Type	Boolean
Definition	Defines whether a QuickTime movie's audio should fade in and/or out with the movie's fade in/out settings (see “**toaFadeIn**” and “**toaFadeOut**” attributes). If omitted, the default value of "**false**" or "**no**" meaning that the audio will not fade (will cut instantly in/out even if a movie fade in/out is specified). If the movie does not fade in or out (both the above attributes are omitted or have values of “**0**”) then this attribute is also irrelevant.
Required For Node Class	None
Optional For Node Class	“**5**” (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.12 “toaInvertFields” Attribute

Attribute Type	Boolean
Definition	Defines whether a QuickTime movie's interlaced fields should be inverted. This may in some rare cases be useful if a movie's field order is incompatible with the Channel's settings (for example, playing a PAL DV formatted file on a standard PAL SD channel). If omitted, the default value of "**false**" or "**no**" meaning that the fields will be left in the order in the QuickTime media. Obviously, this attribute is also irrelevant for progressive display modes.
Required For Node Class	None
Optional For Node Class	“**5**” (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.13 “toaColor” Attribute

Attribute Type	Color
Definition	Specifies the color used to draw the node in the **just:play** and interfaces. Colors may be usefully assigned by the third-party application to indicate the contents or type of **<node>**. For example, playlist nodes might be assigned colors thematically (film, news, ad break etc.). There are no preset colors defined in TOA Solutions, so the third-party application may assign any color to any **<node>**. However, it is recommended that colors be tested within **just:play** or **just:live** to ensure that the results are readable (for example, certain colors may result in text overlays, such as the **<node>**’s name, being difficult for the user to read).
Required For Node Class	None
Optional For Node Class	“**2**” (Playlist), “**5**” (Play)
Required for UI	Yes

8.3.14 "toaContainerType" Attribute

Attribute Type	Index
Definition	Defines how a playlist is started. Valid values are: • **0 for timed,** meaning that the playlist is started at the timecode specified by the "**toaStart**" attribute, even if this meaning that the preceding playlists are "clipped" or cut short by the playlist. The start time is "fixed" and will never be changed by **Just Connect**, even when preceding playlists are changed. Note that this can lead to gaps between playlists, and it is the responsibility of the third-party application to ensure that such gaps do not occur (unless this is desired). • **1 for chained,** meaning that the timecode specified by the "**toaStart**" attribute is "floating" and can be automatically moved up or down by **Just Connect** when the preceding playlists are changed. In this case, the playlist always starts when the preceding playlist ends, so if the preceding playlist's duration is shorted, then this playlist's start time will be moved up in the schedule, and inversely the start time will be moved down in the schedule when the preceding playlist's duration is lengthened.
Required For Node Class	"**2**" (Playlist)
Optional For Node Class	None
Required for UI	Yes

8.3.15 "toaContainerAutoDuration" Attribute

Attribute Type	Boolean
Definition	Defines whether **Just Connect** should automatically set the playlist's duration to the exact duration of all movie play nodes on the playlist's video **<track>**. If omitted, or if specified as "**no**" or "**false**", then **Just Connect** will always use the playlist duration specified by the "**toaDuration**" attribute. This can potentially lead to a playlist being "under" (i.e., too little video material resulting in a gap at the end of the playlist) or "over" (one or more of the movies will be either partially played out ("clipped") or not played out at all ("skipped"). It is the responsibility of the 3rd-party application to ensure that playlists are not "under" or "over", unless this is desired.
Required For Node Class	None
Optional For Node Class	"**2**" (Playlist)
Required for UI	No

8.3.16 "toaContainerLoop" Attribute

Attribute Type	Integer
Definition	Defines how many times a playlist will be looped. The total duration that a playlist is on air is then determined by the **<node>**'s duration as specified by the "**toaDuration**" attribute multiplied by this attribute. For a non-looped playlist, it is, however, required to specify the attribute with a value of "**1**". **Omitting this attribute will lead to the playlist not being played out at all.**
Required For Node Class	"**2**" (Playlist)
Optional For Node Class	None
Required for UI	Yes

8.3.17 "toaTrack" Attribute

Attribute Type	String
Definition	The track ID (e.g., **"v0", "g0". "g1"** etc.) used by **just:live** when the **<node>** is loaded from the **Workbench** to the **Timeline** to specify on which track the **<node>** will run. The value should be valid for the resource type specified by the play **<node>** (i.e., "**v0**" for a video file, "**g0**", "**g1**" etc. for any type of graphic resource). This attribute is not used by **Just Connect** or **Just Out**, so it can be safely omitted if the **just:live** interface is not being used by the user to view the schedules.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	Yes

Only valid for class "**5**" (Play) nodes scheduled for Live Production playout. Please refer to the section on "Live Playout" for further details.

8.3.18 "toaNextAction" Attribute

Attribute Type	Index
Definition	Defines the action taken by **Just Out** when the play **<node>** finishes playing out. The following values are defined: • **0 = do nothing or stop**. Nothing further will be played out on the current **<node>**'s track until the third-party application or **just:live** issues a play command. • **1 = play next**. If another play **<node>** is scheduled on the track on which the current **<node>** is playing, it will be immediately started (i.e., "chained" to the current play **<node>**). • **2 = cue next**. If another play **<node>** is scheduled on the track on which the current **<node>** is playing, its first frame will be cued on the track without playing the **<node>**. To play the next **<node>** the third-party application or **just:live** must issue a play command. • **3 = hold last**. For a video clip, holds the final frame of the video indefinitely until a "**next**" command is issued, after which nothing will be played out (equivalent to the "**stop**" action). • **4 = reload**. The current play **<node>** is "**reloaded**" on the track, meaning that when the third-party application or **just:live** issues a play command for the track, the current **<node>** will play out again, instead of the next scheduled play **<node>** on the track. • **5 = re-cue**. Identical to the "**reload**" end action, but additionally the first frame of the play **<node>**'s media will be displayed on the track. • **6 = hold and cue next.** Identical to the "**hold last**" action, except that the last frame will only be held for the time specified by the "**toaHoldTime**" attribute, after which the next item on the track will automatically be cued. • **7 = hold and play next.** Identical to the "**hold last**" action, except that the last frame will only be held for the time specified by the "**toaHoldTime**" attribute, after which the next item on the track will automatically be played out. If omitted, the default value of "**0**" (do nothing, stop) will be used by **Just Out**.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Only valid for class "**5**" (Play) nodes scheduled for Live Production playout. Please refer to the section on "Live playout" for further details.

8.3.19 "toaTransparentFade" Attribute

Attribute Type	Boolean
Definition	The "**toaFadeIn**" and "**toaFadeOut**" attributes can be used to specify that a movie is faded in from black or out to black. In a live situation with a live SDI signal attached to the video card and downstream keying enabled, it is possible to cross-fade a QuickTime movie from and to the live SDI signal by fading the QuickTime movie from fully transparent to fully opaque and then back again. This can be achieved by specifying this attribute with "**yes**" or "**true**". If omitted, the default value of "**no**" or "**false**" will be used by **Just Out**, meaning that if a fade is specified, it will be from/to black. If no fade in or out is specified for the play **<node>** then this attribute is ignored by **Just Out**.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Only valid for class "**5**" (Play) nodes scheduled for Live Production playout. Optional for play **<nodes>** with QuickTime movie resources, invalid for all other resource types. Please refer to the section on "Live Playout" for further details.

8.3.20 "toaInterfaceMode" Attribute

Attribute Type	Integer
Definition	Defines the default "interface" mode used by **just:live** when the folder is selected in **just:live**. Valid values are: • **0 = list view**. This is the standard workbench table, where play nodes are displayed as a list with columns of data. • **1 = grid view**. This is the grid view where play nodes are displayed as a grid of large thumbnails. If omitted, **just:live** will use the default value of "**0**" (list view) to display the folder's contents.
Required For Node Class	None
Optional For Node Class	"**7**" (Real-time Playlist / Folder)
Required for UI	No

8.3.21 "toaHasCustomInterface" Attribute – DEPRECATED

Attribute Type	Boolean
Definition	Defines whether a play **<node>** has a "**custom interface**" associated with it. If specified as "**yes**" or "**true**" then the **just:live** or **just:play** interfaces will display the custom interface automatically when the **<node>** starts playing out.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	Yes

Valid for class "**5**" (Play) nodes with a **Composition Builder** or Quartz Composer graphic file resource and a custom interface. It can be safely omitted by the 3rd-party application if there is no custom interface for the play node.

8.3.22 "toaPlayLoop" Attribute

Attribute Type	Boolean
Definition	Used in live playout to loop individual play nodes. When this attribute is specified with "**yes**" or "**true**" the play **<node>** will loop endlessly until it is removed from the schedule completely or a command such as "**skip**" is issued on the track on which the **<node>** is playing.
Required For Node Class	"**5**" (Play)
Optional For Node Class	None
Required for UI	Yes

Only valid for class "**5**" (Play) nodes scheduled for Live Production playout. Please refer to the section on "Live Playout" for further details.

8.3.23 "toaVolume" Attribute

Attribute Type	Double
Definition	Specifies a value between "**0.0**" and "**1.0**" at which to play the **<node>**'s audio media. A value of "**0.0**" means that the **<node>**'s audio media is "**mute**" (not heard at all) while a value of "**1.0**" means that the **<node>**'s audio media is played at full volume (exactly as recorded in the media). The volume is scaled linearly, so a value of "**0.5**" means that the **<node>**'s audio media will be played at exactly half the normal volume. If omitted, **Just Out** will use the default value of "**1.0**" (full volume).
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.24 "toaLiveVolume" Attribute

Attribute Type	Double
Definition	Specifies a value between "**0.0**" and "**1.0**" at which to play the live SDI input's audio while the **<node>** is playing. In live playout, **Just Out** will normally play the audio from the live SDI input at full volume if no QuickTime movie is currently playing, and then mute the live SDI input's audio while a movie is playing. However, a user may wish to mix the live input audio with the media's audio, or even play out a movie file with its audio muted and keep the live SDI input's audio playing. A value of "**0.0**" means that the live SDI input's audio media is "**mute**" while a value of "**1.0**" means that the live SDI input's audio media is played out at full volume. The volume is scaled linearly, so a value of "**0.5**" means that the live SDI input's audio media will be played at exactly half the normal volume. If omitted, **Just Out** will use the default value of "**0.0**" (muted). The following combinations are possible: • "**toaVolume**" at "**1.0**" and "**toaLiveVolume**" at "**0.0**". This is the default case: only the audio from the video file will be heard at normal (full) volume. • "**toaVolume**" at "**0.0**" and "**toaLiveVolume**" at "**1.0**". The movie will be seen on the output signal, using the audio from the live SDI input. The movie's audio will not be heard. • "**toaVolume**" at "**1.0**" and "**toaLiveVolume**" at "**0.5**". The movie will be seen and heard at normal (full) volume mixed with the audio from the live SDI input, but with the volume on the live SDI input reduced to 50%.
Required For Node Class	None
Optional For Node Class	5 (Play)
Required for UI	No

Only valid for class "**5**" (Play) nodes scheduled for Live Production playout. Optional for play **<nodes>** with QuickTime movie resources, invalid for all other resource types. Please refer to the section on "Live Playout" for further details.

8.3.25 "toaScheduledDuration" attribute

Attribute Type	Timecode **(excluding date information)**
Definition	This attribute is never sent to **Just Connect** but may be sent back to the third-party applications for class "**5**" (Play) nodes. When present for a play **<node>** this attribute indicates the original scheduled duration for the **<node>** as sent by the third-party application, and it may differ from the actual duration specified by the "**toaDuration**" attribute, which will be the actual duration the **<node>** was on air. The third-party application can use this attribute compared with the "**toaDuration**" attribute to check for nodes that were "**under**" (ended early) or "**over**" (longer on air than originally planned).
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.26 "toaStopFrames" Attribute

Attribute Type	Double Array
Definition	This attribute must be sent for all class "**5**" (Play) nodes with **Composition Builder** or Quartz Composer graphic files that contain stop frames. The size of the array must equal the number of stop frames, and each entry in the array should be the original time in seconds of each stop frame relative to the graphic's timeline in ascending order. This information can then be used by **just:play** when the duration of the play **<node>** is lengthened or shortened to recalculate the stop frame times. Note that this attribute is in addition to the class "**6**" (Trigger) nodes that the play **<node>** must also have (i.e., for each stop frame in a graphic the play **<node>** should have one trigger child **<node>** and one entry in this attribute's array.
Required For Node Class	"**5**" (Play)
Optional For Node Class	None
Required for UI	Yes

8.3.27 "toaStopFrameTime" Attribute

Attribute Type	Timecode **(excluding date information)**
Definition	This attribute must be sent for all class "**6**" (Trigger) nodes, and there must be one such child **<node>** for each stop frame in a **Composition Builder** or Quartz Composer graphic files defined by a class "**5**" (Play) **<node>**. This attribute defines the time in seconds relative to the graphic file's timeline at which the stop frame should be "triggered" or released. This time may be different from the time originally specified in the file if the play **<node>**'s duration has been lengthened, but it should never be earlier than the original value specified in the file as this may lead to unexpected results (e.g., a stop frame not being released at all). The stop frame's original time is specified in the "**toaStopFrames**" attribute on the play **<node>** itself.
Required For Node Class	"**6**" (Trigger)
Optional For Node Class	None
Required for UI	Yes

8.3.28 "toaStopFrameStartTimes" Attribute

Attribute Type	Timecode Array **(including date information)**
Definition	When **Just Out** is playing out a class "**5**" (Play) node that defines a **Composition Builder** or Quartz Composer graphic file with stop frames, each stop frame will be automatically "**held**" indefinitely until either the third-party application or **just:live** issues a command to release the stop frame. Whenever a stop frame is reached and held, **Just Out** will send a "**stopFrameNode**" message and **Just Connect** will forward to all connected clients, including the third-party application. It is the application's responsibility to do two things: • Maintain this attribute for the play **<node>**. For each "**stopFrameNode**" message received, the application should add the timecode sent from **Just Connect** to this timecode array. The array should grow with each stop frame message sent, finally reaching a size equaling the number of stop frames in the graphic file when the final stop frame is released. • Send a response back to **Just Connect** updating the attribute for the **<node>**. Please refer to the "**Communication Protocol**" section for full details on the "**stopFrameNode**" message.
Required For Node Class	"**5**" (Play)
Optional For Node Class	None
Required for UI	Yes

Only valid for class "**5**" (Play) nodes scheduled for Live Playout. Please refer to the section on "Live playout" for further details.

8.3.29 "toaStopFrameEndTimes" Attribute

Attribute Type	Timecode Array **(including date information)**
Definition	This is a sister attribute to the "**toaStopFramesStartTimes**" attribute. When **Just Out** is playing out a class "**5**" (Play) **<node>** that defines a **Composition Builder** or Quartz Composer graphic file with stop frames, each stop frame will be automatically "**held**" indefinitely until either the third-party application or **just:live** issues a command to release the stop frame. Once freed, **Just Out** will release the stop frame and **Just Connect** will forward a "**triggerNode**" message to all connected clients, including the third-party application. It is the application's responsibility to do two things: • Maintain this attribute for the play **<node>**. For each "**triggerNode**" message received, the application should add the timecode sent from **Just Connect** to this timecode array. The array should grow with each stop frame message sent, finally reaching a size equaling the number of stop frames in the graphic file when the final stop frame is released. • Send a response back to **Just Connect** updating the attribute for the **<node>**. Please refer to the "**Communication Protocol**" section for full details on the "**triggerNode**" section.
Required For Node Class	"**5**" (Play)
Optional For Node Class	None
Required for UI	Yes

Only valid for class "**5**" (Play) nodes scheduled for Live Playout. Please refer to the section on "Live playout" for further details.

8.3.30 "toaInfiniteLength" Attribute

Attribute Type	Boolean
Definition	The attribute can be used to flag a "live input" clip as "infinite". Such a clip is given a scheduled duration, but the clip will be played out indefinitely (i.e., with an infinite duration) until the client application send an "**unblockTime**" message. When **Just Out** receives such a message, it will release the "**infinite live input**" clip and immediately play out any following events (video and/or graphic clips), and **Just Connect** will reschedule all playlists following the "**infinite live input**" clip to reflect the actual duration that the clip was on air relative to its original scheduled duration. Please refer to the section on the **<resource>** tag for details on scheduling a live input clip.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.31 "toaScheduledDuration" Attribute

Attribute Type	Timecode
Definition	This timecode attribute can be used to indicate the original scheduled duration of a playlist item (e.g., movie), rather than the actual duration played out. For example, if the "**jump to next clip**" feature is used in **just:play**, the clip currently on air will have this attribute set with the original scheduled duration of the clip, and the "**toaDuration**" attribute will be shortened to the actual duration that the clip was on air. When this attribute is present, it can be used to calculate the difference between the scheduled and actual on air duration for the playlist item, and the result can be displayed as useful information for the user, in this case how much time was "**skipped**" over in the current playlist.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.32 "toaSyncPoint" Attribute

Attribute Type	Timecode
Definition	This timecode attribute can be used to mark a playlist with a "**sync point**". The purpose of a sync point is to display to the user the difference between the actual currently scheduled starting time for the playlist (i.e., the "**toaStart**" attribute) and the "**sync point**" time. For example, a playlist may be scheduled to start at exactly "**19:00:00:00**" and has a "**sync point**" attribute specifying "**19:00:00:00**". The playlist will begin exactly as scheduled by the sync point. However, if the playlist is a chained playlist and something changes in the previous playlists (clips inserted or deleted) then the playlist's start time as indicated by the "**toaStart**" attribute will change, but the sync point timecode will not. This allows to further adjust the preceding playlists to reduce or eliminate the difference between the times and get the playlist back "on time" or "in sync" with the sync point.
Required For Node Class	None
Optional For Node Class	"**2**" (Playlist)
Required for UI	No

8.3.33 "toaEventScript" Attribute - DEPRECATED

Attribute Type	String
Definition	This string attribute contains a full script event in JavaScript that will be compiled and executed by **Just Out** at the timecode specified by the item's "**toaStart**" attribute. A JavaScript event may create and use instances of the ToolsOnAir plugins, such as those to control video routers, GPI switches, HTTP requests, sockets and so on. Full discussion of the plugins available and the possibilities offered by JavaScript events is beyond the scope of this document, please contact ToolsOnAir directly for more information on developing JavaScript events. Note that a play **<node>** with this attribute may also specify a **<resource>** tag with a reference (for example) to the original event script file, but that the script **is not** read from the referenced file but instead always compiled from the text specified by this attribute.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.34 "toaPlaybackSpeed" Attribute

Attribute Type	Index
Definition	This optional attribute can be used to control the playback speed of a video in **Just Out**, using the basic slow-motion feature (frames are duplicated, not blended or interpolated). The following values are valid when defining this attribute: 0 = 100% (normal playback speed) 1 = 75% 2 = 66% 3 = 50% 4 = 33% 5 = 25% Note that the item's duration ("**toaDuration**" attribute) is **not** automatically adjusted for slow motion playback speeds, so when specifying this attribute, the "**toaDuration**" should also be adjusted according to take account of the fact that the item will be playing out more slowly.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.35 "toaPlaybackDeinterlace" Attribute

Attribute Type	Boolean
Definition	This attribute is used in combination with the "**toaPlaybackSpeed**" to indicate whether frames should be deinterlaced when playing back in slow-motion. The default value should this attribute be omitted is "**no**" or "**false**", meaning that frames will not be deinterlaced. When using the slow-motion feature, it is strongly recommended that you test your material before going on air to see whether deinterlacing is necessary or not in slow-motion.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.36 "toaCodec" Attribute

Attribute Type	Integer
Definition	This attribute specifies the codec used to encode the item's video media in the standard Quicktime "4CC" format (four ASCII characters encoded into a 32-bit integer value).
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.37 "toaCodecString" Attribute

Attribute Type	String
Definition	This attribute specifies the full name of the codec used to encode the item's video media. Note that this attribute is purely informational for the user (for example, will be displayed in the **just:play** or **just:live** interfaces) but does not affect the item in any way when playing out.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.38 "toaAspectRatio" Attribute

Attribute Type	Index
Definition	This attribute should always be specified for play items that reference Quicktime movies and specifies the aspect ratio of the video media. The following values are valid: 0 = Unknown aspect ratio (or not one of the following values) 1 = "4:3" aspect ratio 2 = "16:9" aspect ratio Note that the aspect ratio is not necessarily related to the encoded pixels, but rather the aspect ratio that the video media should be ultimately displayed in. For example, video may be encoded in 720x576 pixels ("4:3") but displayed in "16:9" (1024x576 or higher).
Required For Node Class	"**5**" (Play)
Optional For Node Class	None
Required for UI	No

Required for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.39 "toaDisplayWidth" Attribute

Attribute Type	Integer
Definition	This attribute specifies the width in pixels of the video media as it should ultimately be displayed to the user, i.e., taking the aspect ratio into account. For example, a video natively encoded with "720x1024" pixels but with the aspect ratio specified as "16:9" ("**toaAspectRatio**" attribute with a value of "2") should have this attribute specified as "1024" instead of "720".
Required For Node Class	"**5**" (Play)
Optional For Node Class	None
Required for UI	No

Required for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.40 "toaDisplayHeight" Attribute

Attribute Type	Integer
Definition	This attribute specifies the width in pixels of the video media as it should ultimately be displayed to the user, i.e., taking the aspect ratio into account. In most cases, this will only affect the width of the video as specified by the "**toaDisplayWidth**" attribute, but this attribute should also be included for completeness.
Required For Node Class	"**5**" (Play)
Optional For Node Class	None
Required for UI	No

Required for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.41 "toa4to3in16to9Conversion" Attribute

Attribute Type	Index
Definition	This attribute can be used to specify the aspect ratio conversion that should be applied to a video clip that is specified as having a "**4:3**" aspect ratio (by the "**toaAspectRatio**") when the engine output is set to a "**16:9**" aspect ratio. The following values are valid for this attribute: 0 = Full Frame. 1 = Pillar Box. 2 = Scale. 3 = V-Stretch. Note that this attribute is optional and if not specified then the default conversion specified in **Just Out**'s preferences will be applied. By specifying this attribute, the default conversion can be overridden on an item-by-item basis.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Required for play nodes with QuickTime movie resources in "**4:3**" format, invalid for all other resource types.

8.3.42 "toa4to3in4to3Conversion" Attribute

Attribute Type	Index
Definition	This attribute can be used to specify the aspect ratio conversion that should be applied to a video clip that is specified as having a "**4:3**" aspect ratio (by the "**toaAspectRatio**") when the engine output is set to a "**4:3**" aspect ratio. The following values are valid for this attribute: 0 = Play 4:3. 1 = V-Stretch. 2 = Scale. 3 = V-Squeeze to Letterbox. 4 = H-Stretch. Note that this attribute is optional and if not specified then the default conversion specified in **Just Out**'s preferences will be applied. By specifying this attribute, the default conversion can be overridden on an item-by-item basis.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Required for play nodes with QuickTime movie resources in "**4:3**" format, invalid for all other resource types.

8.3.43 "toa16to9in4to3Conversion" attribute

Attribute Type	Index
Definition	This attribute can be used to specify the aspect ratio conversion that should be applied to a video clip that is specified as having a "**16:9**" aspect ratio (by the "**toaAspectRatio**") when the engine output is set to a "**4:3**" aspect ratio. The following values are valid for this attribute: 0 = Full Frame. 1 = Letterbox. 2 = Scale. 3 = H-Stretch. Note that this attribute is optional and if not specified then the default conversion specified in **Just Out**'s preferences will be applied. By specifying this attribute, the default conversion can be overridden on an item-by-item basis.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Required for play nodes with QuickTime movie resources in "**4:3**" format, invalid for all other resource types.

8.3.44 "toa16to9in16to9Conversion" Attribute

Attribute Type	Index
Definition	This attribute can be used to specify the aspect ratio conversion that should be applied to a video clip that is specified as having a "**16:9**" aspect ratio (by the "**toaAspectRatio**") when the engine output is set to a "**16:9**" aspect ratio. The following values are valid for this attribute: 0 = Play 16:9. 1 = H-Stretch. 2 = Scale. 3 = H-Squeeze To Pillarbox. 4 = V-Stretch. Note that this attribute is optional and if not specified then the default conversion specified in **Just Out**'s preferences will be applied. By specifying this attribute, the default conversion can be overridden on an item-by-item basis.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Required for play nodes with QuickTime movie resources in "**4:3**" format, invalid for all other resource types.

8.3.45 "toaAllowResync" Attribute

Attribute Type	Boolean
Definition	This optional attribute can be used to specify whether **Just Out** may re-sync its master and reference timecodes when the playlist finishes playing out. (This assumes that **Just Out** is configured when a reference timecode source such as LTC.) In this case, some frames may be either added at the end of the playlist (after the last video item in the playlist) or some frames clipped from the end of the playlist, depending on whether the master timecode is ahead of or behind the reference timecode. Ideally, assuming that the playlists are relatively short and the drift between the master and reference timecodes small, there should be no difference between the two timecodes after the end of each playlist. The default value if this attribute is not specified for a playlist is "**no**" or "**false**", meaning that the timecodes will not be synchronized by **Just Out** even if there has been drift between the two while the playlist was on air.
Required For Node Class	None
Optional For Node Class	"**2**" (Playlist)
Required for UI	No

8.3.46 "toaFieldOrder" Attribute

Attribute Type	Index
Definition	This attribute specifies the field order of the video material as it was encoded. The following values are valid for this attribute: 0 = Upper Field. 1 = Lower Field. 2 = Progressive. Note that this attribute is purely informational for the user but does not affect the item in any way when playing out.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Optional for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.47 "toaFrameRate" Attribute

Attribute Type	Index
Definition	This attribute specifies the frame rate of the video material as it was encoded. The following values are valid for this attribute: 0 = 23.976 FPS. 1 = 24 FPS. 2 = 25 FPS. 3 = 29.97 FPS. 4 = 30 FPS. 5 = 50 FPS. 6 = 59.94 FPS. 7 = 60 FPS. 8 = Unknown (other) frame rate. Note that this attribute is purely informational for the user but does not affect the item in any way when playing out.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Required for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.48 "toaHoldTime" Attribute

Attribute Type	Timecode
Definition	This attribute should be specified when the "**last action**" specified by the "**toaNextAction**" attribute is either "**Hold and cue next**" or "**Hold and play next**". In this case, this attribute specifies hold long the final frame from the video clip should be held on air for before the next item is automatically either cued or played out.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

Required for play nodes with QuickTime movie resources, invalid for all other resource types.

8.3.49 "toaContainerLocked" Attribute

Attribute Type	Boolean
Definition	Specify "**YES**" to lock the playlist or "**NO**" (the default value if this attribute is omitted) to unlock the playlist. A locked playlist cannot be edited at all, meaning items cannot be added or deleted from the playlist, items within the playlist cannot be edited (e.g., in- or out-point changed, re-ordered), and functions such as split or merge playlists cannot be executed.
Required For Node Class	None
Optional For Node Class	"**2**" (Playlist)
Required for UI	No

8.3.50 "toaTextColor" Attribute

Attribute Type	Color
Definition	Specifies an optional color to use as the default color for the item when displaying it in the **Workbench** and/or **Timeline**. If not specified, then the default text color will be used. This attribute can be useful for highlighting certain items, for example it is used in the **just:play** interface to highlight missing files after refreshing a playlist.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.51 "toaAudioTracks" Attribute

Attribute Type	Integer
Definition	Metadata specifying the number of audio tracks in the video item. This is informational only and will be displayed in the user interface. Please note: if this attribute is not specified then the user will be informed that video has no audio, so it is recommended that this attribute is included for all video items that contain audio.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.52 "toaPaused" Attribute

Attribute Type	Timecode
Definition	If the item has been paused (function only available for **just:live** Channels), then this attribute specifies the timecode at which the item was paused. The timecode is relative to the "**toaStart**" attribute that specifies the timecode at which the item was started, so the difference between the two timecodes specifies the frame at which the item was paused. Once the item is started again, this attribute will be removed.
Required For Node Class	None
Optional For Node Class	"**5**" (Play)
Required for UI	No

8.3.53 “toaPlayStatus” Attribute

Attribute Type	Integer
Definition	For scheduled items in **just:play**, this attribute specifies the item's current playing status. The status is automatically updated by **Just Out** and **Just Connect** as an item is played out, and the status is reflected in **just:play**'s user interface. There can be one of the following values: **0 = Pre-roll:** The item is being pre-rolled in **Just Out**'s buffer and cannot be skipped or deleted at this stage. **1 = Cued:** The item is on-air and cued on the first frame. **2 = Playing:** The item is on-air and playing. **3 = Paused:** The item is on-air and paused. **4 = Finished:** The item was played out and has now finished. Note that if this attribute is not specified, then the item is assumed to be scheduled in the future (neither on-air nor aired).
Required For Node Class	None
Optional For Node Class	“**5**” (Play)
Required for UI	No

8.3.54 “toaAudioMapping” Attribute

<table>
<tr><td>Attribute Type</td><td>Array (double values)</td></tr>
<tr><td>Definition</td><td>
Optional audio mapping that can specify the mapping from the input audio channels in a file to the output audio channels. If specified, should be an array of double values with two values for each audio channel, split into two "sets" of values. The first "set" of values specify the audio mapping for available channels in the input, the second "set" of values specify the audio mapping for unavailable channels in the input.
For example, if you want to specify audio mappings for 2 Channels, the array should have 4 values. The first two values in the array are the "available set" and the second two values are the "not available set". If you want to specify audio mappings for 4 Channels, the array should have 8 values (two sets of 4 values), and so on.
Mappings are specified in output order, and the values are “zero”-based indexes from the input. In all cases, a negative value means "play nothing".
For example, a basic "stereo" mapping would be as follows (array values 0, 1, -1 -1):
<table>
<tr><th>Array index</th><th>Meaning</th><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Output channel “0” mapping if input channel “0” is available.</td><td>0</td><td>On output channel “0”, play input channel “0” if available.</td></tr>
<tr><td>1</td><td>Output channel “1” mapping if input channel “1” is available.</td><td>1</td><td>On output channel “1”, play input channel “1” if available.</td></tr>
<tr><td>2</td><td>Output channel “0” mapping if input channel “0” is not available.</td><td>-1</td><td>On output channel “0”, play nothing if input channel “0” is not available.</td></tr>
<tr><td>3</td><td>Output channel “1” mapping if input channel “1” is not available.</td><td>-1</td><td>On output channel “1”, play nothing if input channel “1” is not available.</td></tr>
</table>
In this scenario, playing a mono input file would result in audio only on the first output Channel (Channel “0”), because Channel “1” is not available on the input, so the "not available" mapping is used and, in this case, it is “-1 (play nothing)”. The following mapping would map a mono input to stereo (array values 0, 0, -1, -1):
<table>
<tr><th>Array index</th><th>Meaning</th><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Output channel “0” mapping if input channel “0” is available.</td><td>0</td><td>On output channel “0”, play input channel “0” if available.</td></tr>
<tr><td>1</td><td>Output channel “1” mapping if input channel “1” is available.</td><td>0</td><td>On output channel “1”, play input channel “0” if available.</td></tr>
<tr><td>2</td><td>Output channel “0” mapping if input channel “0” is not available.</td><td>-1</td><td>On output channel “0”, play nothing if input channel “0” is not available.</td></tr>
<tr><td>3</td><td>Output channel “1” mapping if input channel “1” is not available.</td><td>-1</td><td>On output channel “1”, play nothing if input channel “1” is not available.</td></tr>
</table>
Here, on both output Channels “0” and ”1”, Channel "0” from the input will be selected (assuming it is available), so a mono input would be played on both outputs (assuming stereo output).
</td></tr>
</table>

<table>
<tr><td>Definition (Cont.)</td><td>However, using this mapping an input with stereo would still be played out in "mono" (albeit on both outputs), because Channel “1” on the input is not mapped to any of the outputs and will therefore never be played out. If you wanted to play a stereo file in stereo but still map a mono input to stereo on the output you can use the "input unavailable" values to achieve this (array values 0, 1, -1, 0):

<table>
<tr><th>Array index</th><th>Meaning</th><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Output channel “0” mapping if input channel “0” is available.</td><td>0</td><td>On output channel “0”, play input channel “0” if available.</td></tr>
<tr><td>1</td><td>Output channel “1” mapping if input channel “1” is available.</td><td>1</td><td>On output channel “1”, play input channel “1” if available.</td></tr>
<tr><td>2</td><td>Output channel “0” mapping if input channel “0” is not available.</td><td>-1</td><td>On output channel “0”, play nothing if input channel “0” is not available.</td></tr>
<tr><td>3</td><td>Output channel “1” mapping if input channel “1” is not available.</td><td>0</td><td>On output channel “1”, play input channel “0” if input channel “1” is not available.</td></tr>
</table>

Here input channel 0 will be played on output Channel “0”, and input Channel “1” on output Channel “1” if available, otherwise input Channel “0”. So stereo input (Channels “0” and “1” both available) would play out stereo on outputs “0” and “1”, but in the case of a mono input (only Channel “0” available), then the input Channel “0” would be played on output channels “0” and “1”.

Please note that if this attribute is not specified, then the "standard" audio mapping is used that simply passes all input channels directly to the output channels.</td></tr>
<tr><td>Required For Node Class</td><td>None</td></tr>
<tr><td>Optional For Node Class</td><td>“5” (Play)</td></tr>
<tr><td>Required for UI</td><td>No</td></tr>
</table>

9 Scheduled 24/7 Master Control & Live Production Playout

There are two basic modes of operation supported by **Just Connect** and **Just Out**:

9.1 Scheduled 24/7 Master Control Playout

Scheduled 24/7 Master Control playout allows for items to be scheduled in advance, often up to several days and **Just Connect** feeds each available **Just Out** engine with a rolling schedule for the next **1-2 hours**. The media items in the schedule are loaded "just in time" by **Just Out** and then automatically discarded once played out.

The basic node structure for a 24/7 schedule is as follows:

- Day
  - Playlist
    - Track
      - Play
        - Trigger
      - Play
      - Play
      - Play
    - Track
      - Play
      - Play

In other words, the schedule is split logically into days, each day is split into one or more playlists, each playlist has one or more tracks, and each **<track>** has one or more play nodes. The play nodes define what is played (e.g., a video file or a TOA graphic file) while the track nodes define in which order the play nodes are rendered by **Just Out**.

Playlist nodes and the play nodes that they contain, always define their start times and durations, and are always stored in play order (playlists in the order in which they are played during the day, play nodes in the order they are played in the playlist). Trigger nodes are used to define when triggers are sent to a **Composition Builder** or Quartz Composer graphic file to release (trigger) a stop frame.

9.2 Live Production Playout

For live playout the **<node>** structure is fundamentally different from the 24/7 scheduled Master Control structure due to the very different requirements of the two scenarios. For Live Production playout it is important that relatively few resources are preloaded and therefore available for instant playout, and that the user is free to jump around the rundown. Although a rough running order may be defined for a given event or show, the nature of Live Production playout is much more fluid and free form compared to 24/7 scheduled Master Control playout. Reflecting this, the **<node>** structure for Live Production playout is as follows:

- Real-time root playlist / folder
  - Real-time contents playlist / folder
    - Real-time folder
      - Play
      - Play
    - Real-time folder
      - Play
      - Play
  - Real-time schedule folder
    - Real-time play
    - Real-time play
    - Real-time play

The top levels of this structure are fixed: there is always a root class "**7**" (Real-time playlist / folder) that has two child real-time playlist / folder nodes.

The first of these children is always the "contents" folder, the second the "schedule" folder. The "contents" folder must have at least one child real-time playlist / folder **<node>**, although it may have more than one.

These are logical folders to allow the user to organize the contents or running order of the show, and these are the nodes that the **just:live** interface displays as a list of folders.

Each of these nodes can have one or more play nodes and these nodes represent resources or media (QuickTime movies, **Composition Builder** or Quartz Composer graphic files) that the user can then select for playout.

Once such a play <**node**> is sent to **Just Connect** it will be immediately sent to the related **Just Out** engine which will then preload the media so that it can be played out at any time within the minimum possible latency. To play out one of these nodes, a reference to it is added to the "schedule" folder.

Rather than a copy of the play **<node>**, this special real-time play **<node>** simply refers to the play **<node>** in the "contents" folder by its "**ID**" and the track on which the node should be played out. As such, the "schedule" folder can also be considered a running order. When a client application (**just:live** or a third-party application) sends a command to play the next item on a given track, **Just Out** will take the next available real-time play **<node>** assigned to the given track.

To play out a given media for Live Production playout, a third-party application must take the following steps:

- Either take an existing real-time playlist / folder or create a new folder under the "contents" real-time folder **<node>**.
- In this folder, insert a new play **<node>** with the relevant resource type.
- Insert a new real-time play **<node>** under the "schedule" real-time folder node that references the new play **<node>**'s ID and places it on the desired **<track>**.
- Send a message to play the next available item on the given **<track>**.

The actual XML syntax required for these steps is detailed in the previous sections.

Please note that switching **Just Connect** to the Live Production playout mode is a "one-shot" process - it currently cannot be switched back to 24/7 Master Control playout without quitting and restarting **Just Connect**. Therefore, if you are planning to mix 24/7 Master Control and Live Production playout and want to avoid having to quit and restart **Just Connect** regularly, you will need to carefully plan the playout architecture.

There are two possible ways to work around this situation:

- Have a single instance of **Just Connect** running on the network with two dedicated Channels defined: one for 24/7 Master Control playout and a second for Live Production playout. The third-party application must split the communication between the two channels. All 24/7 Master Control schedules are sent to the 24/7 Master Control Channel and all Live Production information to the Live Production channel.
- Like the above workaround, but with the 2 Channels completely separated on two instances of **Just Connect**.

In both cases, there will be at least two resulting SDI or IP signals: one from the **Just Out** engine assigned to the 24/7 scheduled Master Control playout Channel and another from the **Just Out** engine assigned to the Live Production playout channel. These signals must then be sent to a mixer or other hardware/software so that only the currently relevant signal is sent for broadcast.

10 Playout Communication Protocol

As previously described, the underlying communication protocol used by **Just Connect** is TCP/IP. Specifically, a message is defined as a series of UTF-8 encoded characters terminated by a zero byte. Once a zero byte is detected in the incoming stream, **Just Connect** will then interpret and act on the message. Equally, **Just Connect** will send out messages to all connected client applications (including a third-party application) in the same way, so any application must buffer bytes coming over the socket until a "zero byte" is detected, then interpret the buffered bytes as a "UTF-8" string.

There are two basic classes of messages sent and received by **Just Connect**:

- **Plain text:** These are simple strings with a fixed prefix that may or may not include additional parameters. Any parameters follow the prefix "command" string and a space character.
- **XML:** More complex messages tend to be sent as fully formed XML documents. Such messages will always begin with the "**<**" character (and no plain text message will have this character in the prefix), so any message with this as the first character can be parsed as XML.

In addition, there are three broad types of messages sent and received by **Just Connect**:

- **24/7 Schedule:** These messages relate specifically to 24/7 scheduled Master Control playout.
- **Live:** These messages relate specifically to Live Production playout.
- **General:** These messages either do not relate specifically to 24/7 or live playout or are equally valid for both.

Finally, some messages are sent by the third-party application to **Just Connect,** while others are sent by **Just Connect** to all connected applications. In either case, some messages require a response from the receiver, while others do not. The following sections describe all the messages including their class, type, and response expected, if any.

10.1 "requestNode" Message

Class	XML
Type	24/7 Schedule
Sent By	Client (**just:play** interface, third-party application)
Definition	This message is sent by the client to request the class "**1**" (Day) node for a specific day in the schedule. It is the responsibility of **Just Connect** to manage all day **<nodes>** and the responsibility of the client to manage the day's contents. As such, a client must never send a day **<node>** directly to **Just Connect** but must request it instead. Once received the client application can use the day **<node>**'s "**ID**" to send requests to **Just Connect** to insert playlists for the day, for example. The root tag has the **"date"** attribute to specify the date / timecode of the day being requested. This is in the timecode format described for timecode attributes (including date information). An example request for the day node for "**1.1.2011**" would be as follows: `<requestNode date="1.1.2011 00:00:00:00" />`

Response	The response from **Just Connect** is also XML, as follows:
	
	`<retRequestNode>`
	`  <node class="1" id="xxx" >`
	`  ... schedule XML nodes (playlists etc.) ...`
	`  </node>`
	`</retRequestNode>`
	
	The response is "deep": all Day **<node>**'s descendants (playlists, tracks, play nodes etc.) are included. As such, this message can be used to request the entire current schedule for a given day.

10.2 "requestInsert" Message

Class	XML
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)
Definition	This is a general-purpose node insertion message. It can be used to insert new nodes into the schedule or move existing nodes within the schedule. The **<root>** tag has one required and three optional attributes: • The **"parentId"** attribute is required and specifies the ID of the **<node>** under which the **<node>** being transmitted should be inserted (its parent **<node>**). • The **"beforeId"** attribute optionally defines where in the parent **<node>**'s list of children the new **<node>** should be inserted. When specified, it must be the **ID** of one of the parent **<node>**'s children and the **<node>** will be inserted immediately before this **<node>**. When not specified the new **<node>** is simply appended to the parent (will be the last child **<node>**). • The **"exists"** attribute is an optional Boolean flag ("**F**" for "**false**", "**T**" for "**true**") to indicate whether the **<node>** already exists in the schedule, and it therefore simply being moved in the schedule ("**T**") or is new in the schedule ("**F**"). When omitted, the default value of "**F**" is used, meaning that the **<node>** is new. • The **"setStart"** attribute is an optional Boolean flag ("**F**" for "**false**", "**T**" for "**true**") to indicate whether the **<node>**'s start time should also be automatically calculated. If omitted the default value of "false" means that the start time will not be set and should therefore be included in the message. A value of "true" means that **Just Connect** will attempt to calculate the best possible start time for the **<node>**. This may be useful when inserting a chained playlist into the schedule. Instead of having to calculate the correct start time for the playlist the third-party application can simply set this attribute to "true" and let **Just Connect** calculate the time based on the other playlists already in the schedule. The **<root>** tag should then contain at least one child **<node>** tag to specify the **<node>** or nodes to insert or move in the schedule (multiple nodes are simply represented by multiple child **<node>** tags). If the "exists" attribute is "true" to indicate that existing nodes are being moved, then each child **<node>** tag should simply have the "**id**" attribute set to indicate the **<node>** to be moved in the schedule. For example, the following message... `<requestInsert parentId="xxx" exists="T" setStart="T">   <node id="yyy" /> </requestInsert>` ...would move the existing node with the ID "**yyy**" to the parent **<node>** with the ID "**xxx**", and because the "**beforeId**" attribute is omitted from the message, the **<node>** will be the last child **<node>**. The **<node>**'s start time will also be updated based on the new parent's start time and its child **<nodes>**. If the "**exists**" attribute is "**false**", then the entire contents of the **<node>** (inclusive all attributes and descendant nodes) must be included in the message. For example, the following message... `<requestInsert parentId="xxx" beforeId="yyy" setStart="T">   <node class="2" id="zzz">   ...further XML schedule nodes (e.g. tracks, play nodes)   </node> </requestInsert>` ...will insert a new class "**2**" (Playlist) **<node>** with the ID "**zzz**" under the node with the ID "**xxx**" immediately before the child **<node>** with ID "**yyy**". The **<node>**'s start time will also be updated based on the new parent's start time and its child nodes. The playlist **<node>** should be fully specified in the message (tracks nodes, play nodes etc.).

Response	The response from **Just Connect** is also XML, as follows:
	
	`<retRequestInsert>`
	`  <node id="xxx" ...>`
	`  ... schedule XML nodes (playlists etc.) ...`
	`  </node>`
	`</retRequestInsert>`
	
	The response is "deep": all Day **<node>**'s descendants (playlists, tracks, play nodes etc.) are included. This may be useful in case **Just Connect** has altered anything while inserting the **<node>** into the schedule. For example, if the <**requestInsert**> message specifies the "**setStart**" attribute as "**true**", then the response will include the **<node>**'s "**toaStart**" attribute allowing the sender to read the start time assigned to the **<node>** by **Just Connect**. The response will include one **<node>** tag under the **<root>** tag for each **<node>** tag sent in the original request.

10.3 "requestInsert" Message

Class	XML
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)
	This is a general-purpose **<node>** insertion message. It can be used to insert new nodes into the schedule or move existing nodes within the schedule. The **<root>** tag has one required and three optional attributes: • The **"parentId"** attribute is required and specifies the **ID** of the **<node>** under which the **<node>** being transmitted should be inserted (its parent **<node>**). • The **"beforeId"** attribute optionally defines where in the parent **<node>**'s list of children the new **<node>** should be inserted. When specified, it must be the **ID** of one of the parent **<node>**'s children and the **<node>** will be inserted immediately before this **<node>**. When not specified the new **<node>** is simply appended to the parent (will be the last child **<node>**). • The **"exists"** attribute is an optional Boolean flag ("**F**" for "**false**", "T" for "**true**") to indicate whether the **<node>** already exists in the schedule, and it therefore simply being moved in the schedule ("**T**") or is new in the schedule ("**F**"). When omitted, the default value of "**F**" is used, meaning that the **<node>** is new. • The **"setStart"** attribute is an optional Boolean flag ("**F**" for "**false**", "**T**" for "**true**") to indicate whether the **<node>**'s start time should also be automatically calculated. If omitted the default value of "false" means that the start time will not be set and should therefore be included in the message. A value of "true" means that **Just Connect** will attempt to calculate the best possible start time for the **<node>**. This may be useful when inserting a chained playlist into the schedule. Instead of having to calculate the correct start time for the playlist the third-party application can simply set this attribute to true and let **Just Connect** calculate the time based on the other playlists already in the schedule. The **<root>** tag should then contain at least one child **<node>** tag to specify the **<node>** or nodes to insert or move in the schedule (multiple nodes are simply represented by multiple child **<node>** tags). If the "exists" attribute is "true" to indicate that existing nodes are being moved, then each child **<node>** tag should simply have the "**id**" attribute set to indicate the **<node>** to be moved in the schedule.

Definition	For example, the following message... `<requestInsert parentId="xxx" exists="T" setStart="T">   <node id="yyy" /> </requestInsert>` ...would move the existing node with the ID "**yyy**" to the parent **<node>** with the id "**xxx**", and because the "**beforeId**" attribute is omitted from the message the **<node>** will be the last child **<node>**. The **<node>**'s start time will also be updated based on the new parent's start time and its child nodes. If the "exists" attribute is "false" then the entire contents of the **<node>** (inclusive all attributes and descendant nodes) must be included in the message. For example, the following message... `<requestInsert parentId="xxx" beforeId="yyy" setStart="T">   <node class="2" id="zzz">   ...further XML schedule nodes (e.g. tracks, play nodes)   </node> </requestInsert>` ...will insert a new class "**2**" (Playlist) **<node>** with the **ID "zzz"** under the **<node>** with the **ID "xxx"** immediately before the child **<node>** with **ID "yyy"**. The **<node>**'s start time will also be updated based on the new parent's start time and its child nodes. The playlist **<node>** should be fully specified in the message (tracks nodes, play nodes etc.).
Response	The response from **Just Connect** is also XML, as follows: `<retRequestInsert>   <node id="xxx" ...>   ... schedule XML nodes (playlists etc.) ...   </node> </retRequestInsert>` The response is "deep": all Day **<node>**'s descendants (playlists, tracks, play nodes etc.) are included. This may be useful in case **Just Connect** has altered anything while inserting the **<node>** into the schedule. For example, if the **<requestInsert>** message specifies the "**setStart**" attribute as "true", then the response will include the **<node>**'s "**toaStart**" attribute allowing the sender to read the start time assigned to the **<node>** by **Just Connect**. The response will include one **<node>** tag under the **<root>** tag for each **<node>** tag sent in the original request.

10.4 "requestUpdate" Message

Class	XML
Type	24/7 Schedule
Sent By	Client (**just:play**, 3rd-party application)

Definition	This message is sent by the client to completely update (replace) an existing **<node>** in the schedule with the **<node>** specified in the message. It is a potentially powerful message, able even to completely replace a class "**1**" (Day) **<node>**, thus setting the entire schedule for a day with a single message. The **<root>** tag in this message does not define any attributes. It should have one or more child **<node>** tags, each of which defines the **<node>** to update / replace via the "**id**" attribute. All the **IDs** specified in these **<node>** tags should already exist in the schedule. In addition, the "**class**" attributes specified for the **<node>** tags should also match the classes of the existing nodes in the schedule, otherwise the results will be undefined (e.g., replacing a class "**1**" (Day) **<node>** with a class "**5**" (Play) node would certainly lead to unexpected results). The following XML will replace the existing class "**2**" (Playlist) node with the ID "**xxx**" with the contents specified in the message:
	
	`<requestUpdate>`
	`  <node class="2" id="xxx">`
	`  ... schedule XML nodes (tracks, play nodes etc.)`
	`  </node>`
	`</requestUpdate>`
	
	It is important to note that when updating a class "**1**" (Day) **<node>**, **Just Connect** will automatically assign new unique **IDs** to all descendant nodes (but not the day **<node>** itself). Therefore, the client sending this message must be careful to parse the response from **Just Connect** to update any internal **IDs** with the new **<node>** **IDs** assigned by **Just Connect**. Otherwise, future messages from the client relating to descendants of the day **<node>** will result in errors, as the **IDs** would not be found in the schedule.

Response	The response from **Just Connect** is also XML, as follows:
	
	`<retRequestUpdate>`
	`  <node class="2" id="xxx" >`
	`  ... schedule XML nodes (tracks, play nodes etc.)`
	`  </node>`
	`</retRequestUpdate>`
	
	The response is "deep": all Day **<node>**'s descendants (playlists, tracks, play nodes etc.) are included. Please read the note above in the message definition regarding changed node **IDs** that **Just Connect** may return in its response.

10.5 "requestAttribute" Message

Class	XML
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)

Definition	This message is sent by the client to completely update (replace) an attribute on an existing **<node>** in the schedule. The **<root>** tag in this message defines a single optional attribute:
	• The **"continuous"** attribute is a Boolean flag where "**F**" for "**false**" defines a single, one-shot update of the attribute and "**T**" for "**true**" defines an update that is part of continuous stream of updates for the attributes. A good example of a continuous update is where the attribute is being controlled via a "slider" in the user interface, where the user drags the slider left and right or up and down to quickly and continuously change the value of the attribute. Specifying the continuous flag in this situation is important, as it delays certain updates sent from **Just Connect** to **Just Out**, thus making the updates to the attribute more responsive. Once the user finishes updating the attribute (e.g., releases the mouse button used to drag the slider in the user interface), the client should resend this message one last time with the final attribute value and the "**continuous**" attribute set to "**F**" to ensure that the full update is then sent from **Just Connect** to **Just Out**. If omitted, then the default value of "**F**" is used by **Just Connect**.
	The **<root>** tag in the message should have a single child **<attribute>** tag specifying the attribute to update and one or more **<node>** tags specifying the **<node>**(s) on which the attribute should be updated. Specifying more than one **<node>** tags allow the same attribute to be set on multiple nodes in the schedule with a single message (for example, if the user interface allows the user to select multiple items and then set an attribute, a single "**requestAttribute**" message can be sent by the client). The text of each **<node>** tag defines the **ID** of the **<node>** on which to update the attribute. In each case, a **<node>** with the given **ID** must already exist in the schedule.
	
	The following example sets the "**toaFadeIn**" attribute to a value of "**2.0**" on the nodes with the **IDs** "**xxx**" and "**yyy**":
	
	`<requestAttribute>`
	`  <attribute key="toaFadeIn" type="2">2.0</attribute>`
	`  <node>xxx</node>`
	`  <node>yyy</node>`
	`</requestAttribute>`
	
	Note that this message can also be used to add a new attribute to a given **<node>** as well as updating an existing attribute. If the specified attribute does not exist for one of the given nodes, it will be automatically added to the **<node>**'s attributes.

Response	The response from **Just Connect** is also XML, as follows:
	
	`<retRequestAttribute success="T">`
	`  <attribute key="toaFadeIn" type="2">2.0</attribute>`
	`  <node>xxx</node>`
	`  <node>yyy</node>`
	`</retRequestAttribute>`
	
	• The **"success"** attribute is a Boolean value where "**T**" for "**true**" indicates that the attribute was successfully updated on the requested nodes and "**F**" for "**false**" indicates that the attribute was not updated as requested. This is most likely where the updated value would cause some kind of conflict in the schedule (e.g., setting the duration of a movie play **<node>** to be longer than the movie's natural duration).In this case, the client should check the value of the **<attribute>** tag in the response as this will contain the actual value set on the requested nodes.

10.6 "requestRemoveAttribute" Message

Class	XML
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)
Definition	This message is sent by the client to remove a given attribute from a node in the schedule. The **<root>** tag in this message defines two mandatory attributes: • The **"node"** attribute is the **ID** of the schedule node from which the attribute should be removed. The **<node>** with the given **ID** must already exist in the schedule. • The **"key"** attribute is the key name of the attribute to remove (e.g., "**toaFadeIn**", "**toaFadeOut**"). The following example removes the "**toaFadeIn**" attribute from the nodes with the **ID** "**xxx**": `<requestRemoveAttribute node="xxx" key="toaFadeIn" />` Note that the client sending this message must take care not to remove attributes required for the correct functioning of **Just Connect** and/or **Just Out**. For example, removing the "**toaStart**" or "**toaDuration**" attributes on schedule nodes will lead to undefined behavior, such as **Just Out** playing out nodes at incorrect times.
Response	The response from **Just Connect** is also XML, as follows: `<retRequestRemoveAttribute node="xxx" key="toaFadeIn" />` As such it simply confirms the removed attribute, allowing all clients to update their internal schedule information to reflect the change. Please note that if the **<node>** specified in the original message is not found, then no response will be sent by **Just Connect**.

10.7 "requestDelete" Message

Class	XML
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)

Definition	This message is sent by the client to delete a **<node>** or nodes from the schedule. The **<root>** tag defines no attributes and should have one or more **<id>** tags with the text of each **<id>** tag defining the **ID** of one node in the schedule to be removed.
	The following example removes the nodes **ID** "**xxx**" and "**yyy**":
	`<requestDelete>`
	`  <id>xxx</id>`
	`  <id>yyy</id>`
	`</requestDelete>`
	Note that this is a powerful message as it is possible to remove everything up to and including a class "**1**" (Day) **<node>**. Care should therefore be taken by the client when sending this message to ensure that only nodes that are no longer required in the schedule are removed.

Response	The response from **Just Connect** is also XML, as follows:
	
	`<retRequestDelete>`
	`  <id>xxx</id>`
	`  <id>yyy</id>`
	`</retRequestDelete>`
	
	As such, it simply confirms the removed **<node>**(s), allowing all clients to update their internal schedule information to reflect the change.
	
	Note that if the **<node>** specified in the original message is not found then that **<node>**'s **<id>** tag will not be included in the response sent by **Just Connect**.

10.8 "requestWarnings" Message

Class	XML
Type	24/7 Schedule
Sent By	Client (**just:play**, third-party application)
Definition	This message is sent by the client to request any warnings in the 24/7 schedule such as gaps at the end of a playlist or between two playlists. The **<root>** tag of the message defines two mandatory attributes: • The **"from"** attribute is in the timecode format (including date information) from where the schedule should be checked for warnings. • The **"to"** attribute is in the timecode format (including date information) up to where the schedule should be checked for warnings. The following example checks for warnings in the schedule between "**12:00**" and "**13:00**" on the "**1.1.2011**": `<requestWarnings from="1.1.2011 12:00:00:00" to="1.1.2011 13:00:00:00" />`

Response	The response from **Just Connect** is also XML, as follows:
	
	`<retRequestWarning>`
	`  <warning>Gap at 12:30:00:00</warning>`
	`</retRequestWarning>`
	
	Assuming that a warning is found, it will be sent as the text of the **<warning>** tag. If no warnings are found in the schedule between the time specified by the "from" and "to" attributes the response will simply not include any **<warning>** tags, for example:
	
	`<retRequestWarning />`

10.9 "requestRealTimeContainer" Message

Class	XML
Type	Live
Sent By	Client (**just:live**, third-party application)

Definition	This message has two purposes:
	• Switches **Just Connect** and all **Just Out** engines assigned to the specific channel from "**24/7 scheduled**" playout to "**live**" playout and returns the root class "**7**" (Real-time playlist / folder) **<node>** that the client can then use to manage the live playout (inserting play nodes to the contents / workbench folder, then loading the nodes to the schedule playlist).
	• Preload a live "show" of real-time folder and play nodes.
	
	To simply set **Just Connect** in the "live" mode, send the simple message...
	
	`<requestRealTimeContainer />`
	
	...and to preload a complete live show...
	
	`<requestRealTimeContainer>`
	`  <node class="7" id="xxx">`
	`  ...further Schedule XML nodes`
	`  </node>`
	`</requestRealTimeContainer>`
	
	In this case, the schedule XML nodes must conform to the "live show" structure:
	
	• Class "**7**": Real-time Folder (Root)
	• Class "**7**": Real-time Folder (Contents)
	• Class "**7**": Real-time Folder (Folder)
	• Class "**5**": Play
	• Class "**5**": Play
	• Class "**7**": Real-time Folder (Schedule)
	• Class "**8**": Real-time Play
	• Class "**8**": Real-time Play
	
	Please refer to the "Live Playout" section for full details on the live schedule XML schema.

Response	The response from **Just Connect** is also XML, as follows:
	
	`<retRequestRealTimeContainer>`
	`  <node class="7" id="xxx">`
	`  ...further Schedule XML nodes`
	`  </node>`
	`</retRequestRealTimeContainer>`
	
	Note that the response is "deep": it includes the entire current live schedule. If the client has sent the simple form of the message to create the real-time nodes, then the response will include three class "**7**" nodes: the root real-time playlist with two children. The first child is always the "contents" playlist and the second the "schedule" playlist. The client application can then use the **IDs** of these real-time playlist nodes to manage the live schedule by adding folders and play nodes to the "contents" playlist (preloading contents) and then class "**8**" (Real-time Play) nodes to the "schedule" playlist to schedule the preloaded contents for actual playout.
	Please refer to the "Live Playout" section for full details on the live schedule XML schema.

10.10 "requestFormat" Message

Class	Plain Text
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)
Definition	This message is sent by the client to request the Channel's broadcast format. It is important for the client application to know the Channel's format when calculating timecodes for the schedule. The message is sent without any parameters: `requestFormat`
Response	The response from **Just Connect** is also plain text with a single parameter: `retRequestFormat SD PAL` The string after "**retRequestFormat**" plus a space character specifies the broadcast format. This can be one of the following: • SD PAL • SD NTSC • SD NTSC 23.98 • HD 720p50 • HD 720p60 • HD 1080i25 • HD 1080i29.97 • HD 1080i30 • HD 1080p24 • HD 1080p23.98 • HD 720p24 • HD 720p23.98 • HD 720p59.94

10.11 "requestTracks" Message

Class	Plain Text
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)
Definition	This message is sent by the client to request the Channel's tracks. It may be useful for the client application to know what tracks are defined for the given Channel. For example, the client application may want to allow the user to select which track an item will be rendered on, and obviously, it is important that the user only be presented with a valid list of tracks to select from. The message is sent without any parameters: `requestTracks`

Response	Unusually, the response to this plain text message is sent in XML in the following format:
	
	`<retRequestTracks>`
	`  <track>`
	`  <identifier>g0</identifier>`
	`  <name>Graphic 1</name>`
	`  <master>local._toaengine._tcp.Some-Mac-Pro</master>`
	`  <slave>local._toaengine._tcp.Another-Mac-Pro</master>`
	`  </track>`
	`  <track>`
	`  <identifier>v0</identifier>`
	`  <name>Video 1</name>`
	`  <master>local._toaengine._tcp.Some-Mac-Pro</master>`
	`  <slave>local._toaengine._tcp.Another-Mac-Pro</master>`
	`  </track>`
	`</retRequestTracks>`
	
	The response includes one or more **<track>** tags, each representing a track defined for the Channel. Each **<track>** tag has an **<identifier>** tag where the tag text is the track's **ID** and a **<name>** tag where the tag text is the track's human-readable name (the name that should be presented to the user in any user interface). Finally, the **<track>** tag has a **<master>** tag that defines the Bonjour name of the master system running **Just Out** to render the track and, optionally, a **<slave>** tag that defines the Bonjour name of the slave (redundant) system running **Just Out** to render the track. Multiple tracks can be assigned to the same main (Master) and backup (Slave) **Just Out** systems.
	
	Note that the tracks are defined in reverse-render-order. In other words, the last **<track>** tag will be the first **<track>** rendered (the bottom layer) and is most often the video **<track>**. Further tracks will be rendered bottom-up over another.

10.12 "playtrack" Message

Class	Plain Text
Type	Live
Sent By	Client (**just:live**, third-party application)
Definition	This message is sent by the client to request that the next available item scheduled on the specified **<track>** be played out starting at the next available frame. The next available item is defined as the first class "**8**" (Real-time Play) **<node>** under the "schedule" real-time playlist / folder **<node>** where the "**toaTrack**" attribute matches the specified **<track>**. The message is sent with a single parameter which is the **ID** of the **<track>**. For example, to play the next available video item on the video **<track>** with the identifier "**v0**" the message would be... `playtrack v0` Note that this message will have no effect if an item is already playing on the specified **<track>**. However, if an item is cued on the specified **<track>**, this message will start the item playing.
Response	This message produces no immediate response from **Just Connect**. However, assuming that an item is scheduled for playout on the specified **<track>** then a "**playingNode**" message will be sent by **Just Connect** when the item begins to play out.

10.13 "cuedtrack" Message

Class	Plain Text
Type	Live
Sent By	Client (**just:live**, third-party application)
Definition	This message is sent by the client to request that the next available item scheduled on the specified **<track>** be cued at the next available frame. The next available item is defined as the first class "**8**" (Real-time Play) **<node>** under the "schedule" real-time playlist / folder **<node>** where the "**toaTrack**" attribute matches the specified **<track>**. The message is sent with a single parameter, which is the **ID** of the **<track>**. For example, to cue the next available video item on the video **<track>** with the identifier "**v0**" the message would be... `cuedtrack v0` Note that this message will have no effect if an item is already playing or is cued on the specified **<track>**.
Response	This message produces no immediate response from **Just Connect**. However, assuming that an item is scheduled for playout on the specified **<track>**, then a "**playingNode**" message will be sent by **Just Connect** when the item is cued.

10.14 "nexttrack" Message

Class	Plain Text
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)
Definition	This message is sent by the client to request two actions: • Any item currently playing on the specified **<track>** should be ended at the first possible frame, and... • The next available item scheduled on the specified **<track>** to be played out starting at the next available frame. The next available item is defined as the first class "**8**" (Real-time Play) **<node>** under the "schedule" real-time playlist / folder **<node>** where the "**toaTrack**" attribute matches the specified **<track>**. The message is sent with a single parameter, which is the **ID** of the **<track>**. For example, to play the next available video item on the graphic **<track>** with the identifier "**g0**" the message would be... `nexttrack g0` Please note that the two actions outlined above are completely independent of each other, meaning that one action does not depend on the other action to be valid. For example, sending this message when nothing is currently playing on the specified **<track>** will simply result in the next available item on the **<track>** being played out, as if a "**nexttrack**" command had been sent. Conversely, when an item is currently playing, but no further items are scheduled for the specified **<track>** then the currently playing item will be stopped immediately and nothing played on the **<track>**, as if a "**skiptrack**" message had been sent.
Response	This message produces no immediate response from **Just Connect**. However, one or two messages may be sent by **Just Connect** after a short delay: • Assuming that an item is currently playing on the specified **<track>**, then a "**finishedNode**" message will be sent by **Just Connect** when the item has finished playing out. • Assuming that another item is scheduled for playout on the specified **<track>** then a "**playingNode**" message will be sent by **Just Connect** once the item starts to play out.

10.15 "skiptrack" Message

Class	Plain Text
Type	General
Sent By	Client (**just:play** or **just:live** interfaces, third-party application)

Definition	This message is sent by the client to request that the item currently playing on the specified **<track>** be stopped at the next possible frame. This will result in nothing playing at all on the specified **<track>**.
	The message is sent with a single parameter, which is the **ID** of the **<track>**. For example, to stop playing the current video item on the video **<track>** with the identifier "**v0**" the message would be…
	`skiptrack v0`
	Optionally, it is possible to override the playing item's "**next action**" by sending a different "**next action**" with a command with the following format:
	`skiptrack v0 n`
	Where "**n**" is an integer value specifying the "**next action**" to apply after skipping the currently playing item. The "**next action**" values are identical to those specified by the "**toaNextAction**" attribute, as follows:
	• **0 = do nothing or stop**. Nothing further will be played out on the current **<node>**'s **<track>** until the 3rd-party application or the **just:live** interface issues a play command.
	• **1 = play next**. If another play **<node>** is scheduled on the **<track>** on which the current **<node>** is playing, it will be immediately started (i.e., "chained" to the current play **<node>**).
	• **2 = cue next**. Assuming that another play node is scheduled on the **<track>** on which the current **<node>** is playing, its first frame will be cued on the **<track>** without playing the **<node>**. To play the next **<node>** the third-party application or the **just:live** interface must issue a play command.
	• **3 = hold last**. For a video clip, holds the final frame of the video indefinitely until a "**next**" command is issued, after which nothing will be played out (equivalent to the "**stop**" action).
	• **4 = reload**. The current play **<node>** is "**reloaded**" on the **<track>**, meaning that when the third-party application or the **just:live** interface issues a play command for the **<track>** the current **<node>** will play out again, instead of the next scheduled play node on the **<track>**.
	• **5 = re-cue**. Identical to the "**reload**" end action, but additionally the first frame of the play **<node>**'s media will be displayed on the **<track>**.
	• **6 = hold and cue next.** Identical to the "**hold last**" action, except that the last frame will only be held for the time specified by the "**toaHoldTime**" attribute, after which the next item on the **<track>** will automatically be cued.
	• **7 = hold and play next.** Identical to the "**hold last**" action, except that the last frame will only be held for the time specified by the "**toaHoldTime**" attribute, after which the next item on the **<track>** will automatically be played out.

Response	This message produces no immediate response from **Just Connect**. However, assuming that an item is currently playing on the specified **<track>** then a "**finishedNode**" message will be sent by **Just Connect** when the item has finished playing out.

10.16 "playingNode" Message

Class	XML
Type	General
Sent By	**Just Connect**

Definition	This message is sent by **Just Connect** to all connected clients whenever a class "**5**" (Play) or class "**8**" (Real-time Play) **<node>** begins playing out. The client can use this message to synchronize its internal schedule (by storing the timecode when the item starts playing out) and/or providing status information to the user in a user interface.
	
	The message is formatted as follows:
	
	`<playingNode id="xxx">`
	`  <audioTracks>2</audioTracks>`
	`  <timecode>1.1.2011 12:00:00:00</timecode>`
	`  <nodeUID>yyy</nodeUID>`
	`  <trackID>v0</trackID>`
	`</playingNode>`
	
	The **"id"** attribute on the **<root>** tag identifies the **ID** of the **<node>** being played out. The tag text of the **<audioTracks>** tag defines how many audio tracks the item has (valid for QuickTime movie media items) and the **<timecode>** tag's text defines the starting time of the item in the timecode attribute format (with date information).
	
	The **<nodeUID>** specifies the **UID** of the **<node>** playing (may be different to "**id**" for live nodes, as "**id**" is the "content" **<node> UID** and **<nodeUID>** is the "scheduled" **<node> UID**).
	
	The **<trackID>** tag specifies the identifier of the **<track>** on which the item is playing.

Response	When the client receives this message, and it is currently managing a live playout schedule, it should immediately send a "**requestAttribute**" message back to **Just Connect** reflecting the start time. For example:
	
	`<requestAttribute>`
	`  <attribute key="toaStart" type="7">1.1.2011 12:00:00:00</attribute>`
	`  <node>xxx</node>`
	`</requestAttribute>`
	
	This acknowledges to **Just Connect** that the client has received and accepted the **<node>**'s start time.

10.17 "finishedNode" Message

Class	XML
Type	General
Sent By	**Just Connect**
Definition	This message is sent by **Just Connect** to all connected clients whenever a class "**5**" (Play) or class "**8**" (Real-time Play) node finishes playing out. The client can use this message to synchronize its internal schedule and/or providing status information to the user in a user interface. The **<root>** tag of the message defines three attributes: • The **"id"** attribute defines the ID of the node that finished playing out. • The **"replay"** attribute defines a Boolean value of "**T**" for true if the item will be replayed immediately i.e., it is looped) or "**F**" for "**false**" if the item will not be replayed (it has finished playing out). • The **"skipped"** attribute defines a Boolean value of "**T**" for "**true**" if the item was skipped (finished earlier that originally scheduled due to a user-initiated action) or "**F**" for "**false**" if the item finished playing out normally as originally scheduled. • The **"trackID"** attribute identifies the track on which the **<node>** was playing out. An example message is a follows: `<finishedNode id="xxx" replay="F" skipped="F" trackID="v0"> <timecode>30.4.2012 12:30:00:00</timecode> </finishedNode>`
Response	When the client receives this message and it is currently managing a live playout schedule and the replay attribute is "**F**" for "**false**", it should immediately send the following messages back to **Just Connect**: • A **"requestDelete"** message to remove the class "**8**" (Real-time Play) node referencing the **<node>** that finished playing out (i.e., with the reference **ID** equal to the "**id**" attribute in the original message) from the real-time "schedule" playlist / folder. • A **"requestRemoveAttribute"** message for the "**toaStart**" attribute key on the **<node>** that finished playing out. • A **"requestRemoveAttribute"** message for the "**toaStopFrameStartTimes**" attribute key on the **<node>** that finished playing out. • A **"requestRemoveAttribute"** message for the "**toaStopFrameStopTimes**" attribute key on the **<node>** that finished playing out. For example, the **<finishedNode>** message described above sends the **<node> ID "xxx"**. If the class "**8**" (Real-time Play) node referencing this **<node>** in the real-time "schedule" playlist / folder has the **ID** "**yyy**" the correct response messages would be: `<requestDelete> <id>yyy</id> </requestDelete>` `<requestRemoveAttribute node="xxx" key="toaStart" />` `<requestRemoveAttribute node="xxx" key=" toaStopFrameStartTimes" />` `<requestRemoveAttribute node="xxx" key=" toaStopFrameStopTimes" />`

10.18 "stopFrameNode" Message

Class	XML
Type	General
Sent By	**Just Connect**

Definition	This message is sent by **Just Connect** to all connected clients whenever a class "**5**" (Play) or class "**8**" (Real-time Play) **<node>** with a **Composition Builder** or Quartz Composer graphic file resource reaches a stop frame. The client can use this message to synchronize its internal schedule (by storing the timecode when the stop frame was reached) and/or providing status information to the user in a user interface.
	
	The message is formatted as follows:
	
	`<stopFrameNode id="xxx">`
	`  <timecode>30.4.2012 00:00:00:20</timecode>`
	`</stopFrameNode>`
	
	The **"id"** attribute on the **<root>** tag identifies the **ID** of the **<node>** being played out on which the stop frame was reached. The **<timecode>** tag's text defines the timecode when the stop frame was reached in the timecode attribute format (with date information).

Response	When the client receives this message, and it is managing a live playout schedule it should save the timecode in the **<node>**'s "**toaStopFrameStartTimes**" array and send a "**requestAttribute**" message for the "**toaStopFrameStartTimes**" attribute key back to **Just Connect** to update the attribute. For example:
	
	`<requestAttribute>`
	`  <attribute key="toaStopFrameStartTimes" type="6">`
	`  <array>`
	`  <timecode>00:00:00:20</timecode>`
	`  </array>`
	`  </attribute>`
	`  <node>xxx</node>`
	`</requestAttribute>`

10.19 "triggerNode" Message

Class	XML
Type	General
Sent By	**Just Connect**

Definition	This message is sent by **Just Connect** to all connected clients whenever a class "**5**" (Play) or class "**8**" (Real-time Play) **<node>** with a **Composition Builder** or Quartz Composer graphic file resource is released from a stop frame (i.e., continues playing out). The client can use this message to synchronize its internal schedule (by storing the timecode when the stop frame was released) and/or providing status information to the user in a user interface.
	
	The message is formatted as follows:
	
	`<triggerNode id="xxx">`
	`  <timecode>00:00:00:30</timecode>`
	`</triggerNode>`
	
	The **"id"** attribute on the **<root>** tag identifies the **ID** of the **<node>** being played out on which the stop frame was released. The **<timecode>** tag's text defines the timecode when the stop frame was reached in the timecode attribute format (with date information).

Response	When the client receives this message, and it is managing a live playout schedule, it should save the timecode in the **<node>**'s "**toaStopFrameEndTimes**" array and send a "**requestAttribute**" message for the "**toaStopFrameEndTimes**" attribute key back to **Just Connect** to update the attribute. For example:
	
	`<requestAttribute>`
	`  <attribute key="toaStopFrameEndTimes" type="6">`
	`  <array>`
	`  <timecode>00:00:00:30</timecode>`
	`  </array>`
	`  </attribute>`
	`  <node>xxx</node>`
	`</requestAttribute>`

10.20 "heartbeat" Message

Class	Plain Text
Type	General
Sent By	**Just Connect**
Definition	**Just Connect** will forward a "**heartbeat**" message from all connected **Just Out** engines to the client application (including the third-party application). A heartbeat message is a plain text message with the prefix "**heartbeat**", then a space character and then the current timecode in standard SMPTE timecode format ("**00:00:00:00**"). The client application can use this timecode to synchronize its internal time with **Just Out**'s time and optionally display this information to the user (the current timecode and/or the mere fact the **Just Out** is sending heartbeats to the client). The message is formatted as follows: `heartbeat 12:30:00:30` Note that additional diagnostics information may be appended by **Just Out** after the timecode. This information can be simply ignored by third-party applications.
Response	No response is expected to this message.

10.21 "engineLost" Message

Class	XML
Type	General
Sent By	**Just Connect**
Definition	**Just Connect** will send this message to all connected clients when contact to a given system previously running **Just Out** is lost. The message is formatted as follows: `<engineLost>Some-Mac-Pro</engineLost>` The **<root>** tag text specifies the name of the system running **Just Out** to which contact has been lost.
Response	No response is expected to this message.

10.22 "unblockTime" Message

Class	Plain Text
Type	24/7 Schedule
Sent By	Client (**just:play**, third-party application)
Definition	This command is sent by the client application to release an "**infinite live input**" clip and continue playing any events following the live clip as soon as possible. The message is sent without any parameters, as follows: `unblockTime`
Response	**Just Connect** will respond with at least the "**unblockedTime**" message with the exact timecode at which the time was "**unblocked**" (i.e., at which any events following the "**infinite live input**" clip began playing). This response is also plain text with the timecode (including date information) following the message, as follows: `unblockedTime 1.1.2011 1000` In addition, this message will result in at least one "**finishedNode**" message (for the infinite live input clip itself) and potentially one or more "**playingNode**" messages, one for each event the begins playing out because of the infinite live input clip being released (e.g., a video clip and one or more graphic clips).

11 Playout Example Scenarios

11.1 How do I load a graphic and then set an input port?

To load a **Composition Builder** or Quartz Composer graphic to be played out you will need to send a different set of messages depending on whether you are operating in 24/7 scheduled Master Control or Live Production playout.
In both cases, you first need to send a "**requestInsert**" message containing a class "**5**" (Play) node to define the graphic file you wish to play out.
For 24/7 scheduled Master Control playout you would need to send the play **<node>** as part of a playlist by either inserting the **<node>** into an existing playlist or sending a new playlist. The following example sends a playlist scheduled on "**9.12.2010**" and assumes that the class "**1**" (Day) node for "**9.12.2010**" has the **ID "xxx"**:

```
**********************************************************************************************************************
  <requestInsert parentId="xxx">
    <node id="FCA96D93-5EB5-40DE-8D96-BB0BEFAEAA25" class="2">
      <attribute key="toaColor" type="8">#808080ff</attribute>
      <attribute key="toaName" type="0">New Playlist</attribute>
      <attribute key="toaStart" type="7">9.12.2010 1530000</attribute>
      <attribute key="toaContainerType" type="4">0</attribute>
      <attribute key="toaDuration" type="7">90000</attribute>
      <attribute key="toaContainerLoop" type="1">1</attribute>
      <attribute key="toaContainerAutoDuration" type="3" flags="1">F</attribute>
      <node id="E90DBB38-097D-4D5D-A8E3-BB349AA73C29" class="3" trackId="g0">
        <node id="56202871-D71C-4416-9EEC-4AC3D58B0E43" class="5">
          <attribute key="toaDuration" type="7">75</attribute>
          <attribute key="toaStopFrames" type="6" flags="1">
            <array>
              <double>1.500000</double>
            </array>
          </attribute>
          <attribute key="toaNaturalDuration" type="7" flags="1">75</attribute>
          <attribute key="Component_2_Font_Name_1" type="0" flags="8194"
          name="Font_Name_1_Text">MyriadPro-Regular</attribute>
          <attribute key="Component_2_Font_Size_1" type="2" flags="12290"
          name="Font_Size_1_Text">
            <value>40.000000</value>
            <min>1.000000</min>
          </attribute>
          <attribute key="Component_2_Color" type="8" flags="2"
          name="Color_Text">#000000ff</attribute>
          <attribute key="toaHasCustomInterface" type="3" flags="9">F</attribute>
          <attribute key="toaStart" type="7">9.12.2010 1530000</attribute>
          <attribute key="toaColor" type="8">#00</attribute>
          <attribute key="toaName" type="0">TOA Lower Third 1280x720</attribute>
          <attribute key="Component_2_Justification" type="4" flags="16386"
          name="Justification_Text">
            <value>0</value>
            <max>2</max>
            <values>
              <string>Left</string>
              <string>Center</string>
              <string>Right</string>
            </values>
          </attribute>
          <attribute key="Component_2_Text" type="0" flags="4098" name="Text_Text">Insert Name
          Here</attribute>
          <node id="1DFCD636-0B48-4E7A-9D76-41E17C3F93DE" class="6">
            <attribute key="toaStart" type="7">9.12.2010 1530037</attribute>
            <attribute key="toaStopFrameTime" type="7" flags="1">0</attribute>
          </node>
          <resource type="1">TOA World News/TOA Lower Third
          1280x720.composition/Composition.qtz
          </resource>
        </node>
      </node>
    </node>
  </requestInsert>
**********************************************************************************************************************
```

The actual class "**5**" (Play) node that defines the graphic file is highlighted in bold above. There are several things to note:

- The **<resource>** tag has the "**type**" attribute set to "**1**" to indicate a **Composition Builder** / Quartz Composer resource.
- As the graphic file in the example has a single stop frame the "**toaStopFrames**" attribute is defined with an array of double values, in this case a single value. Additionally, there is a single class "**6**" (Trigger) child **<node>** defined for the stop frame.
- In addition to the standard attributes (those with the "**toa**" prefix in the key attribute) there are also several "**input port**" attributes, one for each input port published by the graphic file. These are the attributes with the "**Component_**" prefix in the key attribute. These attributes always have the "**2**" bit set in the "**flags**" attribute to indicate an input port.
- There are several types of input port defined, such as text (string), color, double, and index.

In terms of 24/7 scheduled Master Control playout, this is enough to bring the graphic file on air at the scheduled time. For Live Production playout you must insert the same play **<node>** into one of the real-time playlist / folders under the "**contents**" real-time playlist / folder. If such a folder has the **ID "xxx"** you would send the following:

```
**********************************************************************************************************************
  <requestInsert parentId="xxx">
    <node id="56202871-D71C-4416-9EEC-4AC3D58B0E43" class="5">
      <attribute key="toaDuration" type="7">75</attribute>
      <attribute key="toaStopFrames" type="6" flags="1">
        <array>
          <double>1.500000</double>
        </array>
      </attribute>
      <attribute key="toaNaturalDuration" type="7" flags="1">75</attribute>
      <attribute key="Component_2_Font_Name_1" type="0" flags="8194"
      name="Font_Name_1_Text">MyriadPro-Regular</attribute>
      <attribute key="Component_2_Font_Size_1" type="2" flags="12290"
      name="Font_Size_1_Text">
        <value>40.000000</value>
        <min>1.000000</min>
      </attribute>
      <attribute key="Component_2_Color" type="8" flags="2"
      name="Color_Text">#000000ff</attribute>
      <attribute key="toaHasCustomInterface" type="3" flags="9">F</attribute>
      <attribute key="toaColor" type="8">#00</attribute>
      <attribute key="toaName" type="0">TOA Lower Third 1280x720</attribute>
      <attribute key="Component_2_Justification" type="4" flags="16386"
      name="Justification_Text">
        <value>0</value>
        <max>2</max>
        <values>
          <string>Left</string>
          <string>Center</string>
          <string>Right</string>
        </values>
      </attribute>
      <attribute key="Component_2_Text" type="0" flags="4098" name="Text_Text">Insert
      Name Here
      </attribute>
      <node id="1DFCD636-0B48-4E7A-9D76-41E17C3F93DE" class="6">
        <attribute key="toaStopFrameTime" type="7" flags="1">0</attribute>
      </node>
      <resource type="1">TOA World News/TOA Lower Third
      1280x720.composition/Composition.qtz
      </resource>
    </node>
  </requestInsert>
**********************************************************************************************************************
```

Please note that the class "**5**" (Play) node itself is identical to that sent in the 24/7 scheduled Master Control playout example except that the "**toaStart**" attributes are missing as the start time will be assigned by **Just Out** when a "**playtrack**" or "**nexttrack**" message is received.

Once loaded for Live Production playout, the **<node>** still needs to be scheduled and then played. If the "schedule" real-time playlist / folder **<node>** has the **ID "yyy"**, the following message would schedule the previously loaded play **<node>**:

```
**********************************************************************************************************************
  <requestInsert parentId="yyy">
    <node id="zzz" class="7" reference="56202871-D71C-4416-9EEC-4AC3D58B0E43"
    trackId="g0" />
  </requestInsert>
**********************************************************************************************************************
```

Now, the play **<node>** with the **ID "56202871-D71C-4416-9EEC-4AC3D58B0E43"** is scheduled for immediate playout. Finally, issue a "**playtrack**" message for the **<track>** on which the **<node>** has been scheduled, in this case "**g0**":

```
playtrack g0
```

The graphic should now be playing out. Regardless of whether 24/7 scheduled Master Control or Live Production playout, once a graphic is on air a message can be sent at any time to change the value of one of the input ports using the "**requestAttribute**" message. For example:

```
**********************************************************************************************************************
  <requestAttribute>
    <attribute key="Component_2_Color" type="8" flags="2"
    name="Color_Text">#ff0000ff</attribute>
    <node>56202871-D71C-4416-9EEC-4AC3D58B0E43</node>
  </requestAttribute>
**********************************************************************************************************************
```

Sets the **<node>**'s attribute with the key "**Component_2_Color**" to the value "**#ff0000ff**" (red color). As this attribute is marked as an input port attribute and the graphic associated with the **<node>** is on air the published input port with the key "**Component_2_Color**" to the value specified in the message.

In this way, the contents of a graphic can be updated at any time, even when the graphic is currently playing on air.

12 Ingest (Capture) Communication Protocol - DEPRECATED

PLEASE NOTE: As of version 5.5 all just:in solutions can directly be accessed via the TOA REST API. Descriptions relating to the former TOA XML Ingest Communication Protocol will be flagged as "DEPRECATED".

The entry-level solutions, namely "just:in mac lite" and "just:in mac lite NDI" were introduced in Fall 2023 and feature the TOA REST API as well.

The description of the Ingest Communication Protocol is nevertheless included in this document for reasons of consistency.

Please visit the official TOA Website to access the **TOA REST API** documentation for the **just:in Capture** solutions:

just:in mac:
https://toolsonair.atlassian.net/wiki/spaces/TST/pages/3940320179/JIM+ToolsOnAir+REST+API+v.6.5

just:in mac lite:
https://toolsonair.atlassian.net/wiki/spaces/TST/pages/3653960356/JIML+ToolsOnAir+REST+API+v.6.x

just:in mac lite NDI:
https://toolsonair.atlassian.net/wiki/spaces/TST/pages/3653993124/JIMLN+ToolsOnAir+REST+API+v.6.x

just:in linux:
https://toolsonair.atlassian.net/wiki/spaces/TST/pages/3662020862/JIL+ToolsOnAir+REST+API+v.6.1

In **just:in** (up to version 5.5) the third-party application talks directly to one or more instances of the **Just In Engine**. Each **Just In Engine** in turn may have one or more Channels, which are identified by the Channel attribute in each message that is sent to or received from the **Just In Engine**. Other than **Just Connect**, **just:in** only uses XML messages for communication.

There are two classes of message:

- Messages sent from the client (**just:in multi-DEPRECATED** or the third-party application) to the **Just In Engine**. In most cases, the **Just In Engine** will send back a response to the client.
- Messages pushed from the **Just In Engine** to the client (**just:in multi-DEPRECATED** or the third-party application).

12.1 "requestChannels" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)
Definition	This message is sent by the client to request all Channels of a given **Just In Engine**. Sent as: `<requestChannels />`

Response	The **Just In Engine** application will respond with one message for each Channel separately as shown below:
	
	`<foundChannel channel="example channel" name="examplehost.local">`
	`  <lock>0</lock>`
	`  <channelIdentifier></channelIdentifier>`
	`</foundChannel>`
	
	• The **"channel"** attribute is the unique identifier for the Channel. The client must use this identifier when sending any messages to **Just In Engine** related to the Channel.
	• The **"name"** attribute defines the human-readable name of the Channel and should be used by the client when displaying information about the channel in its user interface.
	• The text of the **"lock"** tag can be "**0**" or "**1**", where "**0**" means that the Channel is available for use by the client and "**1**" if it is not available (currently locked by another client).
	• The text of the **"channelIdentifier"** contains the current owner of the Channel if it is locked. If the "lock" value is "**0**", the value of "**channelIdentifier**" is empty. A client can use this information to display information to the user (e.g., who currently has the Channel locked).

12.2 "requestLock" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)

Definition	A client sends this message to either request a lock on a Channel (to use the channel to record, for example), or to release a Channel that it had previously locked. The message is sent as follows:
	
	`<requestLock channel="example channel">`
	`  <lock>1</lock>`
	`  <channelIdentifier>example client</channelIdentifier>`
	`</requestLock>`
	
	• The text of the **"lock"** tag should be "**1**" to request that the channel be locked or "**0**" to request the Channel to be released.
	• When locking a Channel, the client should also send a human-readable name of the application locking the Channel in the text of the **"channelIdentifier"** tag. This is the name that will be sent back to clients by the **Just In Engine** is response to the "**requestChannels**" message and may be displayed by clients in their user interfaces.

Response	The **Just In Engine** will respond with the following message:
	
	`<retConfirmLock channel="example channel" name="examplehost.local">`
	`  <lock>1</lock>`
	`  <channelIdentifier>example client</channelIdentifier>`
	`</retConfirmLock>`
	
	• The text of the **"lock"** tag can be "**0**" or "**1**", where "1" means that the channel is locked and "**0**" that the channel is unlocked. The client must interpret the locked status to check whether its request was successful. For example, if the request was to lock a channel and the response has a locked status of "**1**" then the request was successful, whereas a locked status of "**0**" would indicate that the channel was not successfully locked.
	• The text of the **"channelIdentifier"** contains the current owner of the channel if it is locked. If the "lock" value is "**0**", the value of "**channelIdentifier**" is empty. A client can use this information to display information to the user (e.g., who currently has the channel locked).
	
	Note that this response is sent to all connected clients so that each client has an up-to-date list of which Channels are currently locked or unlocked. A client might use this information to display a "live" list of Channels that is automatically updated whenever a Channel is locked or unlocked, even by another client.

12.3 "requestSettingFileNames" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)

Definition	This message asks the **Just In Engine** for all setting files that are available for the given Channel. These setting files are created in **Just In Engine's** System Preferences/Settings pane and are saved in the folder:
	
	`/Library/Application Support/ToolsOnAir/Just In/ *.justin`
	
	For example:
	
	`<requestSettingFileNames channel="example channel" />`
	
	• The **<root>** tag of the message defines the **"channel" attribute,** which must specify the identifier of the Channel for which the presets are being requested. This is the Channel identifier sent as a response to the "**requestChannels**" message.

Response	The **Just In Engine** will respond with the following message:
	
	`<retRequestSettingFileNames channel="example channel" name="examplehost.local">`
	`  <name enabled="0">first.justin</name>`
	`  <name enabled="1">second.justin</name>`
	`  ...`
	`  <name enabled="1">last.justin</name>`
	`</retRequestSettingFileNames>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel and is the same as that sent in the request.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• Each preset file for the Channel is represented by one **<name>** tag in the response. The text of the **<name>** tag is the name of the preset including the "**.justin**" file extension. Client applications wishing to display the presets (e.g., in a list from which the user can select one) should consider stripping this file extension before displaying the name.
	• In addition, each **<name>** tag representing a preset can an **"enabled"** attribute which can have the values "**0**" or "**1**". If the value is "**1**" the setting matches the current **Just In Engine** settings (e.g., the correct frame rate), and is therefore loadable. If the value is "**0**" the setting doesn't correspond with the current **Just In Engine** settings and must not be loaded. It may be displayed to the user in a list but should be disabled so that the user cannot select it.

12.4 "requestLoadSetting" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)
Definition	This message asks the engine to load the specified setting: `<requestLoadSetting channel="example channel">example setting name</requestLoadSetting>` • The **<root>** tag of the message defines the **"channel"** attribute, which must specify the identifier of the Channel for which the presets are being requested. This is the Channel identifier sent as a response to the "**requestChannels**" message. • The text of the **<root>** tag should specify the name of the preset to load **without the ".justin" file extension**. As such, it should match one of the names sent in response to the "**requestSettingFileNames**" messages, except that those names are sent with the "**.justin**" extension and must be stripped before sending the name for this message.

Response	The **Just In Engine** will load the specified setting and when it loads successfully it will respond with the following message:
	
	`<retLoadedSetting channel="example channel" name="examplehost.local" filename="/Library/Application Support/ToolsOnAir/Just In/example setting name.justin">`
	`<codec>DV - PAL</codec>`
	`<aspectratio>0</aspectRatio>`
	`<tvnorm>0</tvnorm>`
	`<audiochannels>0</audiochannels>`
	`<timecodesource>0</timecodesource>`
	`<container>0</container>`
	`<framerate>2500</framerate>`
	`<videowidth>720</videowidth>`
	`<videoheight>576</videoheight>`
	`<TOACompressionComponent>`
	`<name>Apple DV - PAL</name>`
	`<compressionString>1685480304</componentString>`
	`</TOACompressionComponent>`
	`<cliplength>02:00:00:00</cliplength>`
	`</retLoadedSetting>`
	
	• The **"channel"** attribute on the root tag is the identifier of the channel and is the same as that sent in the request.
	• The **"name"** attribute on the root tag is the channel's human-readable name.
	• The **"filename"** attribute on root tag contains the full path of the file from which the presets were loaded.
	• The **<codec>** tag's text is the human-readable name of the codec defined in the present.
	• The **<aspectratio>** tag's text is "0" for 16:9, and "1" for 4:3.
	• The **<tvnorm>** tag's text is "0" for PAL, "1" for NTSC, "2" for HD PAL and "3" for HD NTSC.
	• The **<audiochannels>** tag's text is "0" for 2, "1" for 4 and "2" for 8 audio channels.
	• The **<timecodesource>** tag's text is "0" for Computer Time, "1" for RS422 VTR, "2" for FireWire VTR, 3 for LTC over audio and "4" for Miranda Little Red LTC.
	• The **<container>** tag's text is "0" for QuickTime, "10" for MXF GC, "11" for MXF D10 and "12" for MXF XDCAM.
	• The **<framerate>** tag's text is the preset's frame rate multiplied by 100 (e.g. 2500 for PAL 25 fps).
	• The **<videowidth>** and **<videoheight>** tags' text represent the size in pixels of a video frame.

Response (cont.)	• The **<TOACompressionComponent>** tag is a wrapper of a QuickTime Component. It contains both the name of the codec in the **<name>** tag's text and its component subtype as an integer value in the **<compressionString>** tag's text (the example shows "**dvcp**"). • The **<cliplength>** tag's text is the maximum container length in SMPTE timecode format that is set in **Just In Engine**'s System Preferences/Settings pane.

12.5 "requestDestinationSettingFileNames" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)

Definition	This message is sent by the client to request all destination presets for a given Channel. The destination presets are created in **Just In Engine's** System Preferences/Settings pane. The destination presets are stored in the folder:
	
	`/Library/Application Support/ToolsOnAir/Just In/ *.destination`
	
	The message is sent as follows:
	
	`<requestDestinationSettingFileNames channel="example channel" />`
	
	• The **<root>** tag of the message defines the **"channel"** attribute**,** which must specify the identifier of the Channel for which the presets are being requested. This is the Channel identifier sent as a response to the "**requestChannels**" message.

Response	The **Just In Engine** will respond with:
	
	`<destinationPresets channel="example channel" name="examplehost.local">`
	`<preset>first.destination</preset>`
	`<preset>second.destination</preset>`
	`...`
	`<preset>last.destination</preset>`
	`</destinationPresets>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel and is the same as that sent in the request.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• Each destination preset file for the Channel is represented by one **<preset>** tag in the response. The text of the **<preset>** tag is the name of the preset, including the "**.destination**" file extension. Client applications wishing to display the presets (e.g., in a list from which the user can select one) should consider stripping this file extension before displaying the name.

12.6 "requestLoadDestinationSetting" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)

Definition	This message is sent by the client to request loading the specified destination preset. The message is sent as follows:
	
	`<requestLoadDestinationSetting channel="example channel">`
	`  <presetName>exampleDestinationPreset</presetName>`
	`</requestLoadDestinationSetting>`
	
	• The **<root>** tag of the message defines the **"channel" attribute,** which must specify the identifier of the Channel for which the presets are being requested. This is the Channel identifier sent as a response to the "**requestChannels**" message.
	• The text of the **<root>** tag should specify the name of the destination preset to load **without the ".destination" file extension**. As such, it should match one of the names sent in response to the "**requestDestinationSettingFileNames**" messages, except that those names are sent with the "**.destination**" extension and must be stripped before sending the name for this message.

Response	The **Just In Engine** will load the destination preset and return the following response:
	
	`<returnLoadedDestinationSetting channel="example channel" name="examplehost.local">`
	`  <justinDestinationPreset name="exampleDestinationPreset">`
	`  <hirespath>/path/to/hires</hirespath>`
	`  <lowrespath>/path/to/lowres></lowres>`
	`  <xmlExportpath>/path/to/xmlExport</xmlExportpath>`
	`  </justinDestinationPreset>`
	`  <pathnotwritable>/path/not/writable</pathnotwritable>`
	`  ...`
	`</returnLoadedDestinationSetting>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel and is the same as that sent in the request.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<justinDestinationPreset>** has the "**name**" attribute identifying the name of the destination preset that has been loaded. This name matches the name sent in the request.
	• The **<hirespath>** tag's text defines the path where the full-resolution movie files will be written.
	• The **<lowrespath>** tag's text defines the path where any proxy files will be written, assuming they are enabled for the current preset.
	• The **<xmlExportpath>** tag's text defines the path where any XML metadata files will be written, assuming they are enabled for the current preset.
	• The paths are internally checked for consistency, so when one or more paths are not writable the answer will contain additional **<pathnotwritable>** tags (count depends on how many paths are not writable) with the path(s) that are not writable.

12.7 "requestFilename" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)
Definition	This message asks the **Just In Engine** to write the specified filename, for example: `<requestFilename channel="example channel">exampleName.mov</requestFilename>` • The **<root>** tag of the message defines the **"channel"** attribute, which must specify the identifier of the Channel for which the presets are being requested. This is the Channel identifier sent as a response to the "**requestChannels**" message. • The text of the **<root>** tag specifies the requested filename.

Response	The **Just In Engine** will respond with the following message:
	
	`<retRequestFilename channel="example channel" name="examplehost.local" didChange="F">`
	`exampleName.mov`
	`</retRequestFilename>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel and is the same as that sent in the request.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **"didChange"** on the **<root>** tag can be either "**F**" ("**false**") or "T" ("**true**"). It indicates if the **Just In Engine** changed the requested filename. This may occur if the requested filename already exists at the current destination. In any case, the **<root>** tag's text specifies the filename that will be used by the **Just In Engine** when recording. This will be the same as the filename sent in the request if "**didChange**" is "**F**", or different if "**didChange**" is "**T**".

12.8 "requestRecording" Message

Sent By	Client (**just:in multi-DEPRECATED,** third-party application)
Definition	This message asks the **Just In Engine** to start or stop recording. The message is sent as: `<requestRecording channel="example channel">exampleName.mov</requestRecording>` • The **<root>** tag of the message defines the **"channel"** attribute, which must specify the identifier of the Channel for which the presets are being requested. This is the Channel identifier sent as a response to the "**requestChannels**" message. • The text of the **<root>** tag specifies the requested filename. Just as with the "**requestFilename**" message, the "**requestRecording**" message should include the requested filename for the recording. *The message does not include a recording status (e.g., start or stop recording), as this is simply inferred by the status of the Channel. If the Channel is not currently recording, then this message will start a recording, while if the Channel is already recording, this message will stop the recording. It is up to the client sending the message to appropriately balance this message (once to start recording, again to stop recording).*

Response	The response from the **Just In Engine** will depend on whether the request has started a recording or stopped a recording. When starting a recording, **Just In Engine** will send two responses to this message. The first response will be a "**retRequestFilename**" message to indicate the filename that will be recorded (which may be different to the filename requested). Please refer to the "**requestFilename**" description for full details on this response. Then a second response will be sent:
	
	`<recordingStatus channel="example channel" name="examplehost.local">`
	`<rec>1</rec>`
	`<hours>10</hours>`
	`<minutes>0</minutes>`
	`<seconds>0</seconds>`
	`<frames>0</frames>`
	`</recordingStatus>`
	
	• The **<rec>** tag's text specifies "**1**" to indicate that recording started. The message carries the exact timecode the recording started in the **<hours>**, **<minutes>**, **<seconds>** and **<frames>** tags.
	
	When a recording was stopped the response from the **Just In Engine** is as follows:
	
	`<recordingStatus channel="example channel" name="examplehost.local">`
	`<rec>0</rec>`
	`<hours>10</hours>`
	`<minutes>0</minutes>`
	`<seconds>0</seconds>`
	`<frames>0</frames>`
	`<TOAMovieWriterFrameCount>250<TOAMovieWriterFrameCount>`
	`<TOAMovieWriterClipname>/full/path/to/clip.mov</TOAMovieWriterClipname>`
	`<TOAMovieWriterFirstSplit>1</TOAMovieWriterFirstSplit>`
	`<TOAMovieWriterSplitted>1</TOAMovieWriterSplitted>`
	`<TOAMovieWriterStartTime>***do not use***</TOAMovieWriterStartTime>`
	`<TOAMovieWriterPreviousClipname>previous.mov</TOAMovieWriterPreviousClipname>`
	`<TOAMovieWriterLastSplit>1<TOAMovieWriterLastSplit>`
	`</recordingStatus>`
	
	• The **<rec>**, **<hours>**, **<minutes>**, **<seconds>** and **<frames>** are the same as with the start recording response, except that the text of the **<rec>** tag is "**0**" because the recording was ended.
	• Refer to the section below on "**Movie file chunking**" for details on the **<TOAMovieWriterClipname>**, **<TOAMovieWriterFirstSplit>**, **<TOAMovieWriterSplitted>** and further tags.

12.9 Movie file chunking – DEPRECATED QUICKTIME FORMAT

Due to restrictions in the QuickTime file format, it is not possible to write files longer than a defined duration. Therefore, **just:in** places a restriction on the maximum length (duration) that it will write to a QuickTime movie. Once this limit is reached the video file will be automatically closed, and a new file opened. This process is referred to as "**chunking**".

For example, if the maximum clip length specified in **Just In Engine**'s System Preferences/Settings pane is specified as one hour and a client starts recording on a Channel and then stops after exactly 2.5 hours the result will be three movie files or "chunks": two at exactly an hour in length and a third at half an hour. These chunks can then be joined seamlessly either in **just:play** or in a video editing application such as Final Cut Pro.

Given this chunking process, it is possible that **Just In Engine** will send more than one "**recordingStatus**" message to the client while recording. When each chunk is complete one "**recordingStatus**" message will be pushed to the client (i.e., sent without the client having sent a specific message or request to the **Just In Engine**), and then a final "**recordingStatus**" message will be sent in response to the "**requestRecording**" message sent by the client to stop the recording.

There are several tags in the "**recordingStatus**" message related to the chunking, as follows:

- The **<rec>** tag's text is "**1**" to indicate that the recording is still ongoing and has not yet finished or "**0**" if the recording has finished.
- The **<TOAMovieWriterFrameCount>** tag's text specifies the number of video frames in the chunk just completed.
- The **<TOAMovieWriterClipname>** tag's text specifies the filename of the chunk completed. Obviously, this will change from chunk to chunk.
- The **<TOAMovieWriterFirstSplit>** tag's text is "**1**" to indicate that this is the first chunk in the recording or "**0**" for all following chunks.
- The **<TOAMovieWriterSplitted>** tag's text is "**1**" to indicate that the movie writer itself created this chunk because the maximum clip length was reached, or "**0**" if the chunk split was in response to a user action.
- The **<TOAMovieWriterStartTime>** tag is for internal use only and is not intended for third-party applications.
- The **<TOAMovieWriterPreviousClipname>** tag's text is the filename of the previous chunk in the recording, if the value of the **<TOAMovieWriterFirstSplit>** is "**0**". Otherwise, this is the first chunk, and this tag is not valid.
- The **<TOAMovieWriterLastSplit>** tag's text is "**1**" to indicate that this is the final chunk in the recording or "**0**" if more chunks can be expected.

For example, a status message pushed by the **Just In Engine** to the client when finishing one chunk will be as follows...

```
<recordingStatus channel="example channel" name="examplehost.local">
    <rec>1</rec>
    ...
    <TOAMovieWriterFrameCount>250<TOAMovieWriterFrameCount>
    <TOAMovieWriterClipname>/full/path/to/clip.mov</TOAMovieWriterClipname>
    <TOAMovieWriterFirstSplit>1</TOAMovieWriterFirstSplit>
    <TOAMovieWriterSplitted>1</TOAMovieWriterSplitted>
    <TOAMovieWriterStartTime>***do not use***</TOAMovieWriterStartTime>
    <TOAMovieWriterPreviousClipname>previous.mov</TOAMovieWriterPreviousClipname>
    <TOAMovieWriterLastSplit>0<TOAMovieWriterLastSplit>
</recordingStatus>
```

...while the message for a final chunk in a recording would be...

```
<recordingStatus channel="example channel" name="examplehost.local">
    <rec>0</rec>
    ...
    <TOAMovieWriterFrameCount>250<TOAMovieWriterFrameCount>
    <TOAMovieWriterClipname>/full/path/to/clip#2.mov</TOAMovieWriterClipname>
    <TOAMovieWriterFirstSplit>0</TOAMovieWriterFirstSplit>
    <TOAMovieWriterSplitted>1</TOAMovieWriterSplitted>
    <TOAMovieWriterStartTime>***do not use***</TOAMovieWriterStartTime>
    <TOAMovieWriterPreviousClipname>/full/path/to/clip.mov</TOAMovieWriterPreviou
sClipname>
    <TOAMovieWriterLastSplit>1<TOAMovieWriterLastSplit>
</recordingStatus>
```

12.10 "masterTimecode" Message

Sent By	Server (**Just In Engine**)

Definition	This message is sent every time the timecode changes (i.e., once per video frame) as follows:
	
	`<masterTimecode channel="example channel" name="examplehost.local">`
	`<hours>10</hours>`
	`<minutes>0</minutes>`
	`<seconds>0</seconds>`
	`<frames>0</frames>`
	`<source>0</source>`
	`</masterTimecode>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<source>** tag's text is "**0**" for Computer Time, "**1**" for RS422 VTR, "**2**" for FireWire VTR, "**3**" for LTC over audio and "**4**" for **Miranda Little Red LTC-DEPRECATED**.
	• The **<hours>**, **<minutes>**, **<seconds>** and **<frames>** tags' text define the current timecode.
	
	The client receiving this message may choose to display this timecode in its user interface, for example.

Response	This message does not require a response from the client.

12.11 "previewImage" Message – PARTIALLY DEPRECATED

Sent By	Server (**Just In Engine**)

Definition	This message is sent when a new preview frame is available
	
	`<previewImage channel="example channel" name="examplehost.local">`
	`<previewData>data</previewData>`
	`<is16to9>0</is16to9>`
	`</previewImage>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<previewData>** tag's text (shown as "data" above) is a "**base64**" encoded "**320x240 JPEG**" image. A client receiving this message can send the text through a "base64" decoder and the resulting bytes to a "JPEG" decoder to display the resulting image in its user interface
	• The **<is16to9>** tag's text specifies "**0**" if the preview image has a "**4:3**" aspect ratio or "**1**" if the preview image has a "**16:9**" aspect ratio. Note that in both cases the image will be "**320x240**" but the client receiving this message can choose to respect this value when displaying the preview image.

Response	This message does not require a response from the client.

12.12 "audioMasterLevels" Message

Sent By	Server (**Just In Engine**)

Definition	This message is sent when new audio levels are available as follows:
	
	`<audioMasterLevels channel="example channel" name="examplehost.local">`
	`  <dbValue>-10.923123</dbValue>`
	`  ...`
	`  <dbValue>-inf</dbValue>`
	`</audioMasterLevels>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• For each audio channel, one **<dbValue>** tag will be sent in the message. The tag's text is the value in "**dBu**" from "**-inf**" to "**0**". The number of **<dbValue>** entries is always "**8**" regardless of the number of audio channels currently being captured. The client receiving this message can use the information to provide the user with a visualization of the audio levels coming in on the Channel.

Response	This message does not require a response from the client.

12.13 "canRecord" Message

Sent By	Server (**Just In Engine**)

Definition	This message is sent once a second to indicate whether the **Just In Engine** is ready to record or not.
	
	`<canRecord channel="example channel" name="examplehost.local">`
	`  <rec>0</rec>`
	`</canRecord>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<rec>** tag's text value is "**0**" while the engine is **not** ready to record, or "**1**" if it is ready to record. The client receiving this message may reflect this status by, for example, enabling or disabling a "**record**" button for the given Channel in its user interface.

Response	This message does not require a response from the client.

12.14 "engineMemoryData" Message

Sent By	Server (**Just In Engine**)

Definition	This message is sent once a second to provide the client with information about the system's current memory usage, for example:
	
	`<engineMemoryData channel="example channel" name="examplehost.local">`
	`  <freeRam>6485114880</freeRam>`
	`  <usedRam>2099990528</usedRam>`
	`  <appRam>419320230</freeRam>`
	`</engineMemoryData>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<freeRam>** and **<usedRam>** tags' text are values in bytes. The value of **<freeRam>** is the machine's free and inactive memory, and that of **<usedRam>** the active and wired memory. The sum of the **<freeRam>** and **<usedRam>** values is the total physical memory size of the machine on which the **Just In Engine** is running.
	• The value of **<appRam>** defines how much memory the **Just In Engine** itself is currently using in bytes.

Response	This message does not require a response from the client.

12.15 "engineDiskData" Message – PARTIALLY DEPRECATED

Sent By	Server (**Just In Engine**)

Definition	This message is sent once a second to provide the client with information about available disk space on the storage on which the high-resolution capture path folder is found. The message is as follows:
	
	`<engineDiskData channel="example channel" name="examplehost.local">`
	`  <freeDiskSpace>1456356245</freeDiskSpace>`
	`  <totalDiskSpace>2663432343</totalDiskSpace>`
	`</engineDiskData>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<freeDiskSpace>** and **<totalDiskSpace>** tags' text define values in bytes. Note that Mac OS X 10.6.x uses a "**10^3**" base when calculating disk space (e.g., Finder) whereas the **just:in multi** interface (DEPRECATED) and the **Just In Engine** display these values using a "**2^10**" base. This means that the values specified in TB or GB in Finder and just:in will be differing.

Response	This message does not require a response from the client.

12.16 "engineBufferStatus" Message

Sent By	Server (**Just In Engine**)

Definition	This message is sent once a second to inform the client about the **Just In Engine's** video buffer. It is the same data **Just In Engine** uses in its diagnostics window. The message is as follows:
	
	`<engineBufferStatus channel="example channel" name="examplehost.local">`
	`  <bufferStatus>0.95</bufferStatus>`
	`</engineBufferStatus>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<bufferStatus>** tag's text is a double value between "**0**" and "**1**", where "**0**" indicates a full and "**1**" an empty buffer. If the client receiving this message wants to visualize this metric for the user, then a value of "**0**" is bad (the buffer is full when, for example, the storage system is too slow to write the data) and "**1**" is perfect.

Response	This message does not require a response from the client.

12.17 "dropFrameCount" Message

Sent By	Server (**Just In Engine**)

Definition	This message is sent once a second to inform the client about any frames dropped during recording. It is sent as follows:
	
	`<dropFrameCount channel="example channel" name="examplehost.local">`
	`  <dfCount>0</dfCount>`
	`</dropFrameCount>`
	
	• The **"channel"** attribute on the **<root>** tag is the identifier of the Channel for which this message is being sent.
	• The **"name"** attribute on the **<root>** tag is the Channel's human-readable name.
	• The **<dfCount>** tag's text indicates how many frames were dropped during recording. After stopping the recording, the count will be automatically reset to "**0**", so the value is only valid for the current recording.

Response	This message does not require a response from the client.

13 Ingest (Capture) Example Scenarios

13.1 How do I start / stop recording on a channel?

Assuming that the third-party application is connected to a **Just In Engine** via TCP/IP, the sequence of messages required to start recording on a Channel and then stopping is as follows:

Action	Message sent by client	Response from just:in engine
Discover available channels	**"requestChannels"**	A list of available channels
Lock desired channel	**"requestLock"**	The lock status of the channel
Discover channel's presets	**"requestSettingFileNames"**	A list of preset filenames
Load the desired preset	**"requestLoadSetting"**	Information about the loaded preset
Discover channel's destination presets	**"requestDestinationSettingFileNames"**	A list of destination preset filenames
Load the desired destination preset	**"requestLoadDestinationSetting"**	Information about the loaded preset
Request start recording	**"requestRecording"**	Confirmation on recording status
Request stop recording	**"requestRecording"**	Confirmation on recording status
Release the channel (optionally)	**"requestLock"**	The lock status of the channel

The message flow would be as follows:

Client sends to **Just In Engine**:

```
<requestChannels />
```

...and receives the following message...

```
<foundChannel channel="channel1" name="channel1.local">
  <lock>0</lock>
  <channelIdentifier></channelIdentifier>
</foundChannel>
```

...meaning that the Channel with the identifier **"channel1"** is available and **not** currently locked / in use by another client. If the client then wishes to lock this Channel, the following message would be sent:

```
<requestLock channel="channel1">
  <lock>1</lock>
  <channelIdentifier>My Application</channelIdentifier>
</requestLock>
```

The Channel identifier is the same as that from the "**foundChannel**" message and the **<lock>** tag indicates that the Channel should be locked. The **<channelIdentifier>** tag specifies the name of the third-party application locking the Channel that will be displayed by all other clients on the network. The response, assuming the Channel is successfully locked, would be:

```
<retConfirmLock channel="channel11" name="channel1.local">
  <lock>1</lock>
  <channelIdentifier>My Application</channelIdentifier>
</retConfirmLock>
```

Now that the client has confirmation that the Channel is locked, it can start to use the Channel. First, it must discover the Channel and select a preset by sending the message...

```
<requestSettingFileNames channel="channel1l" />
```

…with the response…

```
<retRequestSettingFileNames channel="channel1" name="channel1.local">
  <name enabled="1">setting.justin</name>
</retRequestSettingFileNames>
```

If the client wishes to load the preset with the name "**setting.justin**" it must then send the message:

```
<requestLoadSetting channel="channel1">setting</requestLoadSetting>
```

Please note that the **".justin"** file extension has been removed from the name sent in the above message. Assuming the preset has been successfully loaded, the response from the **Just In Engine** would be:

```
<retLoadedSetting channel="channel1" name="channel1.local"
filename="/Library/Application Support/ToolsOnAir/Just In/setting.justin">
  <codec>DV - PAL</codec>
  ...
</retLoadedSetting>
```

The client may choose to only display some of the information related to the preset returned to in the response to the user. Next, the client must discover the Channel's destination presets by sending the message:

```
<requestDestinationSettingFileNames channel="channel1" />
```

...to which the response from the **Just In Engine** might be...

```
<destinationPresets channel="channel1" name="channel1.local">
  <preset>setting.destination</preset>
</destinationPresets>
```

If the client wishes to load the preset with the name "**setting.destination**" it must then send the message:

```
<requestLoadDestinationSetting
channel="channel1">setting</requestLoadDestinationSetting>
```

Note that the "**.destination**" file extension has been removed from the name sent in the above message. Assuming the destination preset has been successfully loaded, the response from the **Just In Engine** would be:

```
<returnLoadedDestinationSetting channel="channel1" name="channel1.local">
  <justinDestinationPreset name="setting">
    <hirespath>/path/to/hires</hirespath>
    ...
  </justinDestinationPreset>
</returnLoadedDestinationSetting>
```

The client may choose to only display some of the information related to the destination preset returned in the response to the user.

The Channel is now fully configured and ready to start recording. To start the recording to a video file with the name "**recording.mov**" the client would send the following message:

```
  <requestRecording channel="channel1">recording.mov</requestRecording>
...to which the response would be two messages...
  <retRequestFilename channel="channel1" name="channel1.local" didChange="F">
    recording.mov
  </retRequestFilename>
  <recordingStatus channel="channel1" name="channel1.local">
    <rec>1</rec>
    ...
  </recordingStatus>
```

The first message confirms the filename and indicates that it was not changed by the **Just In Engine**. In other words, the recording will be made to the exact filename requested by the client. The second message indicates that the Channel is now recording.

To stop the recording the client sends another "**requestRecording**" message as follows:

```
  <requestRecording channel="channel1">recording.mov</requestRecording>
```

...to which the response would be...

```
  <recordingStatus channel="channel1" name="channel1.local">
    <rec>0</rec>
    ...
  </recordingStatus>
```

The recording is now finished, and the movie file closed. The Channel itself is again free to initiate another recording at any time. Alternatively, if the client is done with the Channel operation, it should release (unlock) it so that other clients can use the Channel. To do this, it would send the following message:

```
  <requestLock channel="channel1">
    <lock>1</lock>
  </requestLock>
```

14 Related links and additional information:

ToolsOnAir Solutions Overview: https://www.toolsonair.com/wp-content/uploads/overview.pdf

ToolsOnAir Capture Solutions: https://www.toolsonair.com/products/ingest/

ToolsOnAir Playout Solutions: https://www.toolsonair.com/products/playout/

ToolsOnAir Solutions Datasheets: https://www.toolsonair.com/wp-content/uploads/datasheet.pdf

ToolsOnAir User Manuals: https://toolsonair.atlassian.net/wiki/spaces/TST

ToolsOnAir Tutorial Videos: https://www.youtube.com/c/ToolsonairBroadcastEngineering

ToolsOnAir Support T&Cs (Online): https://www.toolsonair.com/support-tcs/