UDP Commands User Manual

Model No. ET-FMP50
ET-FMP20
ET-SBFMP10

Panasonic

(2026-03)

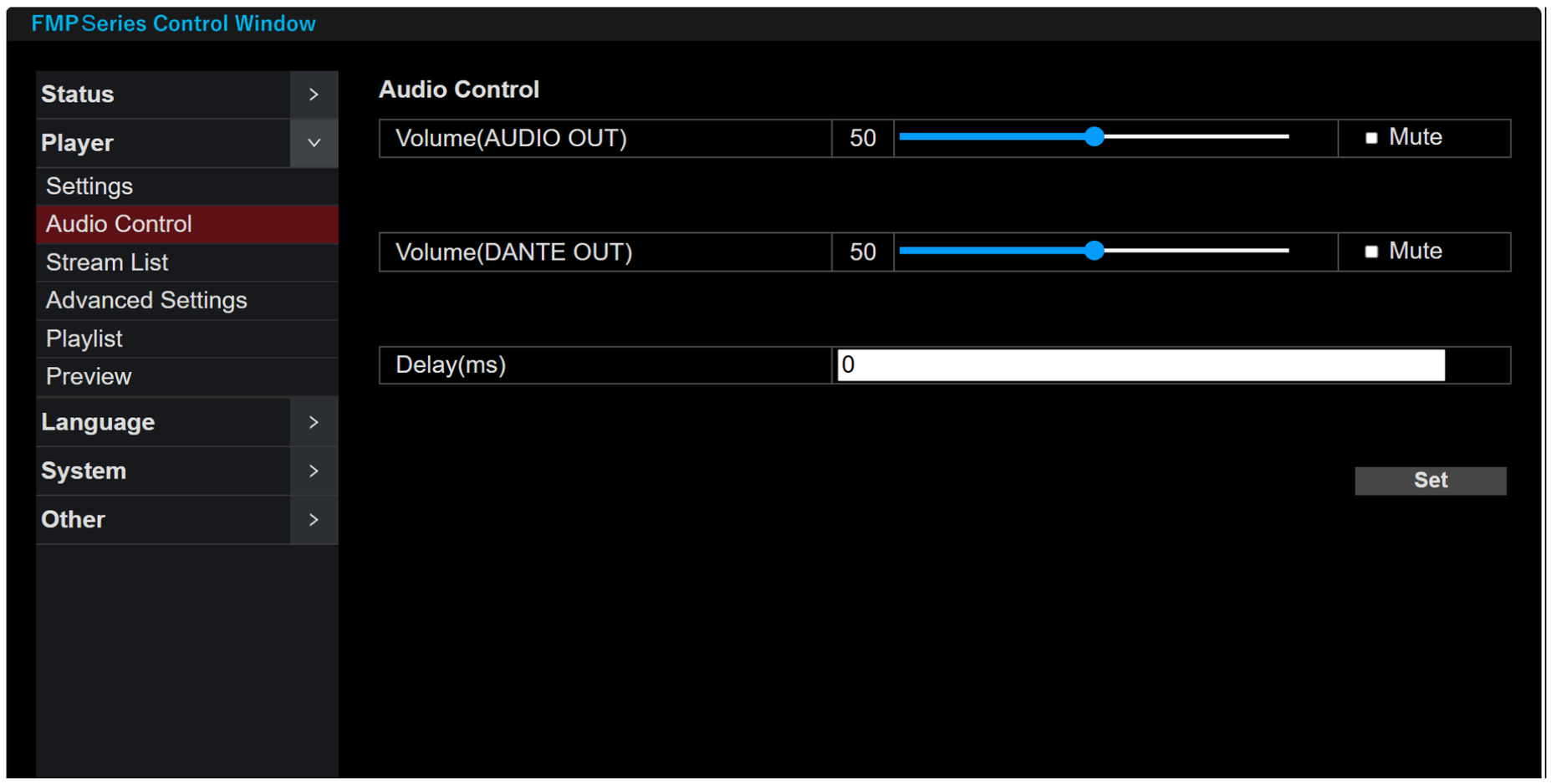
■How to Use UDP Commands (Setting Control)

●Overview

Various settings made on the web page of the Media Processor (FMP) can be performed using UDP command communication.

※ : Some playback and file operation related settings are not supported.

Example) When performing volume changes or mute operations on the web page,
it is possible to change the volume or mute without opening the web page using UDP commands.



●UDP Command Specifications

FMP Connection Conditions

The FMP is running and waiting on the following socket.

Address: Address set in FMP

Listening port number: 65430

Source address: INADDR_ANY

Maximum send/receive size: 65536 bytes

List of Operation Commands

Command	Description
SetPlayerProperty	**Set player information**
SetAudioProperty	**Set audio information**
SetAdvancedProperty	**Set advanced settings information**
PlayStream	**Start NDI streaming playback**
StopStream	**Stop streaming playback**
SetAutoPlayProperty	**Set auto-playback stream information**
SetPlaylistSyncProperty	**Set playlist synchronization information**
SetLanguage	**Set display language**
SetHostName	**Set hostname**
SetLEDProperty	**Set POWER LED information**
SetNetworkProperty	**Set network information**
SetTimeZone	**Set time zone**
SetDatetimeProperty	**Set date/time information**
SetAccountProperty	**Change Web account information**
execInitialize	**Initialize system**
Reboot	**Restart system**

List of Information Commands

Comand	Description
GetPlayerState	**Get player playback status**
GetPlayerProperty	**Get player information**
GetAudioProperty	**Get audio information**
GetAdvancedProperty	**Get advanced settings information**
GetStreamList	**Get list of NDI streams**
GetAutoPlayProperty	**Get auto-playback stream information**
GetPlaylistSyncProperty	**Get playlist synchronization information**
GetLanguage	**Get display language**
GetHostName	**Get hostname**
GetLEDProperty	**Get POWER LED information**
GetNetworkProperty	**Get network information**
GetDatetimeProperty	**Get date/time information**

Player Transfer Command

The following setting control commands can be used for the "Playback control commands" and "PinP control commands" described later.
To operate, add a space after the ProxyPlayer command, and then specify the command you want to use (playback control, PinP control).

Command	Description
ProxyPlayer	**Proxy to Player** Transfers UDP commands to the Player (Schedule / NDI / Timeline). The UDP port on the Player side is "65432". Returns responses from the Player as is.

Request Example:

ProxyPlayer GetSystemTime CurrentTimeVss=1715237197598000

Response:

- UDP transfer failed (UDP transfer failed)　※If Player is not running, UDP transmission timeout (10s)
- Return value from Player

OK CurrentTimeVss=1715237197598000&CurrentTimeFmp=1715237196004200

Common Errors

Invalid command (NG Unknown command)
Web account does not exist (NG Account not set up)

	Get player playback status

Command	**GetPlayerState**

Description

Command to get player playback status.

Request

Parameter	Always	Description	Remarks
None			

Request Example

GetPlayerState

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
Player	○	Selected player (1:Schedule 2:NDI(NDI Decoder) 3:Timeline)	
ByVss	○	Timeline started with VSS (0:non-VSS start 1:VSS start)	
Content		Content name / signal information (NDI)	Not set when stopped ※ encodeURIComponent

Return Example

On Sccess

OK Player=1&ByVss=0&Content=abc.mp4

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "=".

On Failure

None

	Get player information

Command	**GetPlayerProperty**

Description

Command to get player information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetPlayerProperty
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	Always [OK] is returnd
Player	○	Selected player (1:Schedule 2:NDI(NDI Decoder) 3:Timeline)	
OutputMode	○	Output mode(1, 4)	
VideoSignal	○	Video signal (0:AUTO 1:3840x2160/60p 2:3840x2160/50p 3:1080/60p 4:1080/50p)	
AudioOutput	○	Audio output (0:HDMI OUT 1:AUDIO OUT 2:DANTE)	
Screen1IPAddress		Screen 1 IP address	Only when output mode is "4"
Screen2IPAddress		Screen 2 IP address	Only when output mode is "4"
Screen3IPAddress		Screen 3 IP address	Only when output mode is "4"
Screen4IPAddress		Screen 4 IP address	Only when output mode is "4"

Return Example

On Success

```
OK Player=2&OutputMode=4&VideoSignal=0&AudioOutput=0&Screen1IPAddress=192.168.0.1&
                        Screen2IPAddress=192.168.0.2&Screen1IPAddress=192.168.0.3&Screen4IPAddress=192.168.0.4
```

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "=".

On Failure

None

	Set player information

Command	**SetPlayerProperty**

Description

Command to set player information.

※Processing may take up to 30 seconds for a response.

Request

Parameter	Always	Description	Remarks
Player	○	Player (1:Schedule 2:NDI(NDI Decoder) 3:Timeline)	
OutputMode	○	Output mode(1, 4)	Not applicable for SDM (SBFMP10)
VideoSignal	○	Video signal (0:AUTO 1:3840x2160/60p 2:3840x2160/50p 3:1080/60p 4:1080/50p)	
AudioOutput	○	Audio output (0:HDMI OUT 1:AUDIO OUT 2:DANTE)	
Screen1IPAddress		Screen 1 IP address	Required if output mode is "4"
Screen2IPAddress		Screen 2 IP address	Required if output mode is "4"
Screen3IPAddress		Screen 3 IP address	Required if output mode is "4"
Screen4IPAddress		Screen 4 IP address	Required if output mode is "4"

Request Example

```
SetPlayerProperty Player=1&OutputMode=4&VideoSignal=2&AudioOutput=2&Screen1IPAddress=192.168.0.1&
                  Screen2IPAddress=192.168.0.2&Screen1IPAddress=192.168.0.3&Screen4IPAddress=192.168.0.4
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Get audio information

Command	**GetAudioProperty**

Description

Command to get audio information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetAudioProperty
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
HdmiVolume	○	Volume(0 - 100)	Not applicable for SDM (SBFMP10)
HdmiMute	○	Mute(true, false)	Not applicable for SDM (SBFMP10)
AnalogVolume	○	Volume(0 - 100)	
AnalogMute	○	Mute(true, false)	
DanteVolume	○	Volume(0 - 100)	
DanteMute	○	Mute(true, false)	
Delay	○	Audio delay time(0 - 171)	

Return Example

On Success

```
OK HdmiVolume=50&HdmiMute=false&AnalogVolume=50&AnalogMute=false&DanteVolume=50&DanteMute=false&Delay=17

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
```

On Failure

None

	Set audio information

Command	**SetAudioProperty**

Description

Command to set audio information.

Request

Parameter	Always	Description	Remarks
HdmiVolume	○	Volume(0 - 100)	Not applicable for SDM (SBFMP10)
HdmiMute	○	Mute(true, false)	Not applicable for SDM (SBFMP10)
AnalogVolume	○	Volume(0 - 100)	
AnalogMute	○	Mute(true, false)	
DanteVolume	○	Volume(0 - 100)	
DanteMute	○	Mute(true, false)	
Delay	○	Audio delay time(0 - 171)	

Request Example

SetAudioProperty HdmiVolume=50&HdmiMute=false&AnalogVolume=50&AnalogMute=false&DanteVolume=50&DanteMute=false&Delay=17

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Get advanced settings information

Command	**GetAdvancedProperty**

Description

Command to get advanced settings information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetAdvancedProperty
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
AntiAliasing	○	Anti-aliasing (0.00:OFF 0.25:Low 0.50:High)	

Return Example

On Success

```
OK AntiAliasing=0.50

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
```

On Failure

None

	Set advanced settings information

Command	**SetAdvancedProperty**

Description

Command to set advanced settings information.

Request

Parameter	Always	Description	Remarks
AntiAliasing	○	Anti-aliasing (0.00:OFF 0.25:Low 0.50:High)	

Request Example

SetAdvancedProperty AntiAliasing=0.50

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Get NDI stream list

Command	**GetStreamList**

Description

Command to get NDI stream list.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetStreamList
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
StreamNum	○	Number of streams	
Source1		1st source	※ encodeURIComponent
Status1		1st playback status(0:Stopped 1:Playing)	
Source2		2nd source	※ encodeURIComponent
Status2		2nd playback status(0:Stopped 1:Playing)	
...			
SourceN		Nth source	※ encodeURIComponent
StatusN		Nth playback status(0:Stopped 1:Playing)	

Return Example

On Success

```
OK StreamNum=2&Source1=xxx&Status1=0&Source2=yyy&Status2=1


※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
```

On Failure

```
NG Not Support

※：NG may be due to the following reasons.
   ・NDI not selected, VSS Timeline is active (Not Support)
```

	Start NDI streaming playback

Command	**PlayStream**

Description

Command to start NDI streaming playback.

※Processing may take up to 10 seconds for a response.

Request

Parameter	Always	Description	Remarks
Source	○	Source	※ decodeURIComponent

Request Example

PlayStream Source=xxx

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
- Invalid parameter
- NDI not selected, VSS Timeline active (Not Support)
- Source does not exist (Execution failed)

	Stop streaming playback

Command	**StopStream**

Description

Command to stop streaming playback.

Request

Parameter	Always	Description	Remarks
None			

Request Example

StopStream

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Not Support

※：NG may be due to the following reasons.
・NDI not selected, VSS Timeline active (Not Support)

	Get auto-playback stream information

Command	**GetAutoPlayProperty**

Description

Command to get auto-playback stream information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetAutoPlayProperty
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
AutoPlayStatus	○	Auto-playback enabled status(true, false)	
AutoPlaySource		Auto-playback stream source	※ encodeURIComponent

Return Example

On Success

```
OK AutoPlayStatus=true&AutoPlaySource=PSDCD-KK-6444%20(Test%20Pattern)


※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
```

On Failure

```
NG Not Support

※：NG may be due to the following reasons.
  ・NDI not selected, VSS Timeline active (Not Support)
```

	Set auto-playback stream information

Command	**SetAutoPlayProperty**

Description

Command to set auto-playback stream information.

Request

Parameter	Always	Description	Remarks
AutoPlayStatus	○	Auto-playback enabled status(true, false)	
AutoPlaySource	○	Source	※ decodeURIComponent

Request Example

SetAutoPlayProperty AutoPlayStatus=true&AutoPlaySource=PSDCD-KK-6444%20(Test%20Pattern)

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
- Invalid parameter
- NDI not selected, VSS Timeline active (Not Support)
- Source does not exist (Execution failed)

	Get playlist synchronization information

Command	**GetPlaylistSyncProperty**

Description

Command to get playlist synchronization information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetPlaylistSyncProperty
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
PlaylistSync	○	Playlist synchronization information (true, false)	
PlayContolSync	○	Playback control synchronization information (true, false)	

Return Example

On Success

```
OK PlaylistSync=false&PlayContolSync=false


※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
```

On Failure

None

	Set playlist synchronization information

Command	**SetPlaylistSyncProperty**

Description

Command to set playlist synchronization information.

Request

Parameter	Always	Description	Remarks
PlaylistSync	○	Playlist synchronization information (true, false)	
PlayContolSync	○	Playback control synchronization information (true, false)	

Request Example

SetPlaylistSyncProperty PlaylistSync=false&PlayContolSync=true

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Get display language

Command	**GetLanguage**

Description

Command to get display language.

Request

Parameter	Always	Description	Remarks
None			

Request Example

GetLanguage□

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
Language	○	Display language(0:English 1:Japanese)	

Return Example

On Success

OK Language=0

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

None

	Set display language

Command	**SetLanguage**

Description

Command to set display language.

Request

Parameter	Always	Description	Remarks
Language	○	Display language(0:English 1:Japanese)	

Request Example

SetLanguage Language=0

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Get hostname

Command	**GetHostName**

Description

Command to get hostname.

Request

Parameter	Always	Description	Remarks
None			

Request Example

GetHostName

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
Hostname	○	Hostname	

Return Example

On Success

OK Hostname=NAME3821

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

None

	Set hostname

Command	**SetHostName**

Description

Command to set hostname.

Request

Parameter	Always	Description	Remarks
Hostname	○	Hostname	

Request Example

```
SetHostName Hostname=NAME3821
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Get POWER LED information

Command	**GetLEDProperty**

Description

Command to get POWER LED information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

GetLEDProperty

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
Mode	○	Mode(0:Nomal 1:Off)	
Notification	○	Notification(0:Disable 1:Enable)	

Return Example

On Success

OK Mode=0&Notification=1

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

None

	Set POWER LED information

Command	**SetLEDProperty**

Description

Command to set POWER LED information.

Request

Parameter	Always	Description	Remarks
Mode	○	Mode(0:Normal 1:Off)	
Notification	○	Notification(0:Disable 1:Enable)	

Request Example

SetLEDProperty Mode=0&Notification=1

※：If Mode is "0:Normal", Notification will be "1:Enable" regardless of the set value.

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Get network information

Command	**GetNetworkProperty**

Description

Command to get network information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetNetworkProperty
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
DHCP	○	DHCP(true:ON, false:OFF)	
IPAddress		IP address	Set if DHCP is "false"
SubnetMask		Subnet mask	Set if DHCP is "false"
Gateway		Default gateway	Set if DHCP is "false"
DNS		DNS	

Return Example

On Success

```
OK DHCP=false&IPAddress=192.168.0.1&SubnetMask=255.255.255.0&Gateway=192.168.0.2&DNS=8.8.8.8
```

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

None

	Set network information

Command	**SetNetworkProperty**

Description

Command to set network information.

※Processing may take up to 20 seconds for a response.

Request

Parameter	Always	Description	Remarks
DHCP	○	DHCP(true:ON, false:OFF)	
IPAddress		IP address	Set if DHCP is "false"
SubnetMask		Subnet mask	Set if DHCP is "false"
Gateway		Default gateway	Set if DHCP is "false"
DNS		DNS	

Request Example

SetNetworkProperty DHCP=false&IPAddress=192.168.0.1&SubnetMask=255.255.255.0&Gateway=192.168.0.2&DNS=8.8.8.8

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：In case of IP change, response cannot be received due to network reboot.
※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
- Invalid parameter
- Setting failed (Execution failed)

	Get date/time information

Command	**GetDatetimeProperty**

Description

Command to get date/time information.

Request

Parameter	Always	Description	Remarks
None			

Request Example

GetDatetimeProperty

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
TimezoneLocation	○	Time zone region	
TimezoneCity	○	Time zone city	
CurrentTime	○	Current time(YYYYMMDDhhmmss)	
SyncProtocol	○	Synchronization protocol (0:NTP 1:SoftwareSync)	
DomainNo		Domain number(0 - 127)	Only when synchronization protocol is SoftwareSync
OrderPs		Primary / Secondary (0:Primary 1:Secondary)	Only when synchronization protocol is SoftwareSync
NTPSynchronization		NTP synchronization(true:ON false:OFF)	When synchronization protocol is NTP when synchronization protocol is SoftwareSync, Primary / Secondary is Primary
NTPServer		NTP server	Only when NTP synchronization is ON

Return Example

On Success

If synchronization protocol is NTP:
OK TimezoneLocation=Asia&TimezoneCity=Tokyo&CurrentTime=20250925101010&SyncProtocol=0&NTPSynchronization=true&NTPServer=192.168.0.1

'If synchronization protocol is SoftwareSync:
OK TimezoneLocation=Asia&TimezoneCity=Tokyo&CurrentTime=20250925101010&SyncProtocol=1&DomainNo=0&OrderPs=0
&NTPSynchronization=true&NTPServer=192.168.0.1

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

None

	Set time zone

Command	**SetTimeZone**

Description

Command to set time zone.

Request

Parameter	Always	Description	Remarks
TimezoneLocation	○	Time zone region	
TimezoneCity	○	Time zone city	

Request Example

SetTimeZone TimezoneLocation=Africa&TimezoneCity=Abidjan

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Set date/time information

Command	**SetDatetimeProperty**

Description

Command to set date/time information.

Request

Parameter	Always	Description	Remarks
SyncProtocol	○	Synchronization protocol (0:NTP 1:SoftwareSync)	
DomainNo		Domain number(0 - 127)	Required if synchronization protocol is SoftwareSync
OrderPs		Primary / Secondary (0:Primary 1:Secondary)	Required if synchronization protocol is SoftwareSync
NTPSynchronization		NTP synchronization(true:ON false:OFF)	Required if synchronization protocol is NTP Required if synchronization protocol is SoftwareSync, Primary / Secondary is Primary
NTPServer		NTP server	Required if NTP synchronization is ON
Date		Date(YYYY/MM/DD)	If NTP synchronization is OFF, ignore set content
Time		Time(hh:mm:ss)	If NTP synchronization is OFF, ignore set content

Request Example

```
SetDatetimeProperty SyncProtocol=0&NTPSynchronization=true&NTPServer=192.168.0.1
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter

	Change Web account information

Command	**SetAccountProperty**

Description

Content | Command to change Web account information.

Request

Parameter	Always	Description	Remarks
CurrentUserName	○	Current username	
CurrentPassword	○	Current password	
UserName		Username	Error if username and password are all unset.
Password		Password	

Request Example

SetAccountProperty CurrentUserName=a&CurrentPassword=b&UserName=c

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
- Invalid parameter
- Authentication error (Authentication failed)
- Same as existing account (The same value is entered)

	System initialization

Command	**ExecInitialize**

Description

Command to initialize the system.

※Processing may take up to 20 seconds for a response.
※The system will automatically restart after executing this command.

Request

Parameter	Always	Description	Remarks
All	○	All settings (0:Full initialization 1:Partial Initialization)	
Player		Player settings(true, false)	If all settings are "1:Partial initialization", at least one item must be set to true. If "All" / "Network" is set, the IP address will be initialized.
Geometory		Geometry correction settings(true, false)	
Network		Network settings(true, false)	
Content		Content data(true, false)	

Request Example

ExecInitialize All=0
ExecInitialize All=1&Player=true&Geometory=false&Network=false&Content=false

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter
・Authentication error (Authentication failed)

	System reboot

Command	Reboot

Description
Command to reboot the system.

Request			
Parameter	Always	Description	Remarks
None			

Request Example
Reboot

Return Value			
Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd

Return Example
On Success
OK ※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
On Failure
None

●Sample code for Setting Control（python）

This is a sample code for playback control.

Please change 'DEFAULT_FMP_IP' and 'BROADCAST_IP' according to the IP address of the FMP you are using (local IP).

```
import socket
import time
import re

# --- Constant Settings ---
M_SIZE = 1024  # Receive buffer size

# FMP address and target port number (default for single FMP)
# Please change according to your actual FMP address
DEFAULT_FMP_IP = '192.168.0.11'  # Default IP for single FMP (Change according to your environment)
FMP_PORT = 65430  # Port number for configuration control commands

# Broadcast address (usually .255 for IPv4 /24 subnet)
# Adjust if your subnet mask is different
BROADCAST_IP = '192.168.0.255'  # Example broadcast IP for a /24 subnet (Change according to your environment)

# --- UDP Socket Initialization ---
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)  # Enable broadcast mode for this socket
# Default timeout for most operations (can be overridden by command-specific settings)
DEFAULT_SOCKET_TIMEOUT = 10
sock.settimeout(DEFAULT_SOCKET_TIMEOUT)

# --- Global Variables for Configuration State ---
# Add command-specific global variables as needed

# --- Helper Functions for sending commands ---
def send_command_single_fmp(target_ip, command_str, timeout_seconds=DEFAULT_SOCKET_TIMEOUT):
    """
    Sends the specified UDP command string to a single FMP and receives a response.
    Returns a dictionary with FMP IP as key and response string as value.
    'timeout_seconds' can be used to set a command-specific timeout.
    """
    target_address = (target_ip, FMP_PORT)
    print(f"Sending to {target_address}: {command_str}")
    try:
        sock.settimeout(timeout_seconds)  # Set command-specific timeout
        sock.sendto(command_str.encode('utf-8'), target_address)
        rx_message, addr = sock.recvfrom(M_SIZE)
        decoded_message = rx_message.decode(encoding='utf-8')
        print(f"Received from {addr}: {decoded_message}")
        return {addr[0]: decoded_message}  # Return as a dictionary for consistency
    except socket.timeout:
        print(f"Error: Socket timeout ({timeout_seconds}s) when sending to {target_address}. FMP might not be responding.")
        return {target_ip: "ERROR: TIMEOUT"}
    except Exception as e:
        print(f"Error sending/receiving data to {target_address}: {e}")
        return {target_ip: f"ERROR: EXCEPTION ({e})"}
    finally:
        sock.settimeout(DEFAULT_SOCKET_TIMEOUT) # Reset to default timeout

def send_command_broadcast(command_str, timeout_seconds=3, expected_fmp_count=None):
    """
    Sends the specified UDP command string via broadcast and collects responses from multiple FMPs.
    Returns a dictionary with FMP IP as key and response string as value for each FMP.
    """
    broadcast_address = (BROADCAST_IP, FMP_PORT)
```

```
print(f"Sending broadcast to {broadcast_address}: {command_str}")
responses = {}

# Temporarily set a shorter timeout for collecting multiple responses in broadcast mode
sock.settimeout(0.1)
start_time = time.time()
try:
    sock.sendto(command_str.encode('utf-8'), broadcast_address)
    while time.time() - start_time < timeout_seconds:
        try:
            rx_message, addr = sock.recvfrom(M_SIZE)
            decoded_message = rx_message.decode(encoding='utf-8')
            print(f"Received from {addr}: {decoded_message}")
            responses[addr[0]] = decoded_message  # Store response with IP as key
            if expected_fmp_count is not None and len(responses) >= expected_fmp_count:
                print(f"Collected {len(responses)} responses, which is the expected count.")
                break
        except socket.timeout:
            pass
        except Exception as e:
            print(f"Error receiving data in broadcast loop: {e}")
            break
    if not responses:
        print("No FMP responses received within the timeout period.")
    return responses
except Exception as e:
    print(f"Error sending broadcast or setting up socket: {e}")
    return {}
finally:
    sock.settimeout(DEFAULT_SOCKET_TIMEOUT) # Reset to default timeout


def execute_command(send_mode, command_str, target_ip=None, timeout_seconds=DEFAULT_SOCKET_TIMEOUT):
    """Wrapper function to execute command based on send_mode, with configurable timeout."""
    if send_mode == '1':  # Single FMP
        if target_ip is None:
            target_ip = input(f"Enter target FMP IP address (default: {DEFAULT_FMP_IP}): ") or DEFAULT_FMP_IP
        return send_command_single_fmp(target_ip, command_str, timeout_seconds)
    elif send_mode == '2':  # Broadcast
        broadcast_timeout = float(input("Enter broadcast response collection timeout in seconds (e.g., 3.0): ") or 3.0)
        expected_count_input = input("Enter expected number of FMPs (leave blank if unknown): ")
        expected_count = int(expected_count_input) if expected_count_input else None
        return send_command_broadcast(command_str, timeout_seconds=broadcast_timeout, expected_fmp_count=expected_count)
    else:
        print("Invalid send mode. Please choose 1 or 2.")
        return {}

def parse_response_params(response_str):
    """
    Parses FMP's OK response string into a dictionary of parameters.
    Example: "OK Player=1&ByVss=0&Content=abc.mp4" -> {"Player": "1", "ByVss": "0", "Content": "abc.mp4"}
    """
params = {}
if not response_str.startswith("OK"):
    return params

# Remove "OK " prefix and split by "&"
param_pairs = response_str[3:].split('&')
for pair in param_pairs:
    if '=' in pair:
        key, value = pair.split('=', 1)
        params[key] = value
    else:
        params[pair] = True  # For parameters like "-d" (debug option) without value
```

```
    return params

# --- Configuration Command Handling Functions ---

def handle_get_player_state(send_mode, target_ip):
    """P.5 Get Player Playback Status GetPlayerState"""
    print("\n--- GetPlayerState ---")
    command = "GetPlayerState"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get player state.")

def handle_get_player_property(send_mode, target_ip):
    """P.6 Get Player Information GetPlayerProperty"""
    print("\n--- GetPlayerProperty ---")
    command = "GetPlayerProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get player property.")

def handle_set_player_property(send_mode, target_ip):
    """P.7 Set Player Information SetPlayerProperty
    *This process may take up to 30 seconds for a response."""
    print("\n--- SetPlayerProperty ---")

    player = ""
    while player not in ['1', '2', '3']:
        player = input("Player (1:Schedule, 2:NDI, 3:Timeline, required): ")
        if player not in ['1', '2', '3']:
            print("Invalid value for Player. Please enter 1, 2, or 3.")

    output_mode = ""
    while output_mode not in ['1', '4']:
        output_mode = input("OutputMode (1, 4, required): ")
        if output_mode not in ['1', '4']:
            print("Invalid value for OutputMode. Please enter 1 or 4.")

    video_signal = ""
    while not video_signal: # VideoSignal is required
        video_signal = input("VideoSignal (0:AUTO, 1:3840x2160/60p, ..., required): ")
        if not video_signal:
            print("VideoSignal is required.")

    audio_output = ""
    while audio_output not in ['0', '1', '2']:
        audio_output = input("AudioOutput (0:HDMI OUT, 1:AUDIO OUT, 2:DANTE, required): ")
        if audio_output not in ['0', '1', '2']:
            print("Invalid value for AudioOutput. Please enter 0, 1, or 2.")
```

```
params = []
params.append(f"Player={player}")
params.append(f"OutputMode={output_mode}")
params.append(f"VideoSignal={video_signal}") # Add VideoSignal
params.append(f"AudioOutput={audio_output}")

# IP address settings are required only if OutputMode is 4 (mandatory)
if output_mode == '4':
    screen1_ip = ""
    while not screen1_ip:
        screen1_ip = input("Screen1IPAddress (required for OutputMode=4): ")
        if not screen1_ip: print("Screen1IPAddress is required.")

    screen2_ip = ""
    while not screen2_ip:
        screen2_ip = input("Screen2IPAddress (required for OutputMode=4): ")
        if not screen2_ip: print("Screen2IPAddress is required.")

    screen3_ip = ""
    while not screen3_ip:
        screen3_ip = input("Screen3IPAddress (required for OutputMode=4): ")
        if not screen3_ip: print("Screen3IPAddress is required.")

    screen4_ip = ""
    while not screen4_ip:
        screen4_ip = input("Screen4IPAddress (required for OutputMode=4): ")
        if not screen4_ip: print("Screen4IPAddress is required.")

    params.append(f"Screen1IPAddress={screen1_ip}")
    params.append(f"Screen2IPAddress={screen2_ip}")
    params.append(f"Screen3IPAddress={screen3_ip}")
    params.append(f"Screen4IPAddress={screen4_ip}")

    command = "SetPlayerProperty " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip, timeout_seconds=30)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetPlayerProperty command sent successfully.")
    else:
        print("Failed to send SetPlayerProperty command.")


def handle_get_audio_property(send_mode, target_ip):
    """P.8 Get Audio Information GetAudioProperty"""
    print("\n--- GetAudioProperty ---")
    command = "GetAudioProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get audio property.")

def handle_set_audio_property(send_mode, target_ip):
    """P.9 Set Audio Information SetAudioProperty"""
    print("\n--- SetAudioProperty ---")

    hdmi_volume = ""
    while not hdmi_volume.isdigit() or not (0 <= int(hdmi_volume) <= 100): # HdmiVolume is required
        hdmi_volume = input("HdmiVolume (0-100, required): ")
```

```
    if not hdmi_volume.isdigit() or not (0 <= int(hdmi_volume) <= 100):
        print("Invalid value for HdmiVolume. Please enter a number between 0 and 100.")

hdmi_mute = ""
while hdmi_mute not in ['true', 'false']:
    hdmi_mute = input("HdmiMute (true/false, required): ")
    if hdmi_mute not in ['true', 'false']:
        print("Invalid value for HdmiMute. Please enter true or false.")

analog_volume = ""
while not analog_volume.isdigit() or not (0 <= int(analog_volume) <= 100): # AnalogVolume is required
    analog_volume = input("AnalogVolume (0-100, required): ")
    if not analog_volume.isdigit() or not (0 <= int(analog_volume) <= 100):
        print("Invalid value for AnalogVolume. Please enter a number between 0 and 100.")

analog_mute = ""
while analog_mute not in ['true', 'false']: # AnalogMute is required
    analog_mute = input("AnalogMute (true/false, required): ")
    if analog_mute not in ['true', 'false']:
        print("Invalid value for AnalogMute. Please enter true or false.")

dante_volume = ""
while not dante_volume.isdigit() or not (0 <= int(dante_volume) <= 100):
    dante_volume = input("DanteVolume (0-100, required): ")
    if not dante_volume.isdigit() or not (0 <= int(dante_volume) <= 100):
        print("Invalid value for DanteVolume. Please enter a number between 0 and 100.")

dante_mute = ""
while dante_mute not in ['true', 'false']:
    dante_mute = input("DanteMute (true/false, required): ")
    if dante_mute not in ['true', 'false']:
        print("Invalid value for DanteMute. Please enter true or false.")

delay = input("Delay (0-171, leave blank for no change): ") # Not required

params = []
params.append(f"HdmiVolume={hdmi_volume}") # Add HdmiVolume
params.append(f"HdmiMute={hdmi_mute}")
params.append(f"AnalogVolume={analog_volume}") # Add AnalogVolume
params.append(f"AnalogMute={analog_mute}") # Add AnalogMute
params.append(f"DanteVolume={dante_volume}")
params.append(f"DanteMute={dante_mute}")
if delay: params.append(f"Delay={delay}")

if not params: # Error if no parameters are set
    print("No parameters provided. Exiting.")
    return

command = "SetAudioProperty " + "&".join(params)
responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetAudioProperty command sent successfully.")
    else:
        print("Failed to send SetAudioProperty command.")

def handle_get_advanced_property(send_mode, target_ip):
    """P.10 Get Advanced Settings GetAdvancedProperty"""
    print("\n--- GetAdvancedProperty ---")
    command = "GetAdvancedProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
```

```
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get advanced property.")

def handle_set_advanced_property(send_mode, target_ip):
    """P.11 Set Advanced Settings SetAdvancedProperty"""
    print("\n--- SetAdvancedProperty ---")
    anti_aliasing = ""
    while anti_aliasing not in ['0.00', '0.25', '0.50']:
        anti_aliasing = input("AntiAliasing (0.00:OFF, 0.25:Low, 0.50:High, required): ")
        if anti_aliasing not in ['0.00', '0.25', '0.50']:
            print("Invalid value for AntiAliasing. Please enter 0.00, 0.25, or 0.50.")

    params = []
    params.append(f"AntiAliasing={anti_aliasing}")

    command = "SetAdvancedProperty " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetAdvancedProperty command sent successfully.")
    else:
        print("Failed to send SetAdvancedProperty command.")

def handle_get_stream_list(send_mode, target_ip):
    """P.12 Get NDI Stream List GetStreamList"""
    print("\n--- GetStreamList ---")
    command = "GetStreamList"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                stream_num = int(parsed_params.get('StreamNum', 0))
                print(f"  StreamNum: {stream_num}")
                for i in range(1, stream_num + 1):
                    source = parsed_params.get(f'Source{i}', 'N/A')
                    status = parsed_params.get(f'Status{i}', 'N/A')
                    print(f"    Source{i}: {source}, Status{i}: {status}")
    else:
        print("Failed to get stream list.")

def handle_play_stream(send_mode, target_ip):
    """P.13 Start NDI Streaming PlayStream
    *This process may take up to 10 seconds for a response."""
    print("\n--- PlayStream ---")
    source = input("Source (required): ")
    if not source:
        print("Source is required. Exiting.")
        return
    command = f"PlayStream Source={source}"
    responses = execute_command(send_mode, command, target_ip, timeout_seconds=10)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print(f"PlayStream command sent successfully for source: {source}.")
    else:
        print(f"Failed to send PlayStream command for source: {source}.")

def handle_stop_stream(send_mode, target_ip):
    """P.14 Stop Streaming Playback StopStream"""
    print("\n--- StopStream ---")
```

```
    command = "StopStream"
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("StopStream command sent successfully.")
    else:
        print("Failed to send StopStream command.")

def handle_get_autoplay_property(send_mode, target_ip):
    """P.15 Get AutoPlay Stream Information GetAutoPlayProperty"""
    print("\n--- GetAutoPlayProperty ---")
    command = "GetAutoPlayProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get auto play property.")

def handle_set_autoplay_property(send_mode, target_ip):
    """P.16 Set AutoPlay Stream Information SetAutoPlayProperty"""
    print("\n--- SetAutoPlayProperty ---")
    autoplay_status = ""
    while autoplay_status not in ['true', 'false']:
        autoplay_status = input("AutoPlayStatus (true/false, required): ")
        if autoplay_status not in ['true', 'false']:
            print("Invalid value for AutoPlayStatus. Please enter true or false.")

    autoplay_source = ""
    while not autoplay_source: # AutoPlaySource is required
        autoplay_source = input("AutoPlaySource (required): ")
        if not autoplay_source:
            print("AutoPlaySource is required.")

    params = []
    params.append(f"AutoPlayStatus={autoplay_status}")
    params.append(f"AutoPlaySource={autoplay_source}")

    command = "SetAutoPlayProperty " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetAutoPlayProperty command sent successfully.")
    else:
        print("Failed to send SetAutoPlayProperty command.")

def handle_get_playlist_sync_property(send_mode, target_ip):
    """P.17 Get Playlist Sync Information GetPlaylistSyncProperty"""
    print("\n--- GetPlaylistSyncProperty ---")
    command = "GetPlaylistSyncProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get playlist sync property.")
```

```
def handle_set_playlist_sync_property(send_mode, target_ip):
    """P.18 Set Playlist Sync Information SetPlaylistSyncProperty"""
    print("\n--- SetPlaylistSyncProperty ---")
    playlist_sync = ""
    while playlist_sync not in ['true', 'false']:
        playlist_sync = input("PlaylistSync (true/false, required): ")
        if playlist_sync not in ['true', 'false']:
            print("Invalid value for PlaylistSync. Please enter true or false.")

    play_control_sync = ""
    while play_control_sync not in ['true', 'false']:
        play_control_sync = input("PlayControlSync (true/false, required): ")
        if play_control_sync not in ['true', 'false']:
            print("Invalid value for PlayControlSync. Please enter true or false.")

    params = []
    params.append(f"PlaylistSync={playlist_sync}")
    params.append(f"PlayControlSync={play_control_sync}")

    command = "SetPlaylistSyncProperty " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetPlaylistSyncProperty command sent successfully.")
    else:
        print("Failed to send SetPlaylistSyncProperty command.")

def handle_get_language(send_mode, target_ip):
    """P.19 Get Display Language GetLanguage"""
    print("\n--- GetLanguage ---")
    command = "GetLanguage"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get language.")

def handle_set_language(send_mode, target_ip):
    """P.20 Set Display Language SetLanguage"""
    print("\n--- SetLanguage ---")
    language = ""
    while language not in ['0', '1']:
        language = input("Language (0:English, 1:Japanese, required): ")
        if language not in ['0', '1']:
            print("Invalid value for Language. Please enter 0 or 1.")

    params = []
    params.append(f"Language={language}")

    command = "SetLanguage " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetLanguage command sent successfully.")
    else:
        print("Failed to send SetLanguage command.")

def handle_get_host_name(send_mode, target_ip):
    """P.21 Get Host Name GetHostName"""
    print("\n--- GetHostName ---")
```

```
    command = "GetHostName"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get host name.")

def handle_set_host_name(send_mode, target_ip):
    """P.22 Set Host Name SetHostName"""
    print("\n--- SetHostName ---")
    hostname = ""
    while not hostname:
        hostname = input("Hostname (required): ")
        if not hostname:
            print("Hostname is required.")

    params = []
    params.append(f"Hostname={hostname}")

    command = "SetHostName " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetHostName command sent successfully.")
    else:
        print("Failed to send SetHostName command.")

def handle_get_led_property(send_mode, target_ip):
    """P.23 Get POWER LED Information GetLEDProperty"""
    print("\n--- GetLEDProperty ---")
    command = "GetLEDProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get LED property.")

def handle_set_led_property(send_mode, target_ip):
    """P.24 Set POWER LED Information SetLEDProperty"""
    print("\n--- SetLEDProperty ---")
    mode = ""
    while mode not in ['0', '1']:
        mode = input("Mode (0:Normal, 1:Off, required): ")
        if mode not in ['0', '1']:
            print("Invalid value for Mode. Please enter 0 or 1.")

    notification = ""
    while notification not in ['0', '1']:
        notification = input("Notification (0:Disable, 1:Enable, required): ")
        if notification not in ['0', '1']:
            print("Invalid value for Notification. Please enter 0 or 1.")

    params = []
    params.append(f"Mode={mode}")
```

```
    params.append(f"Notification={notification}")

    command = "SetLEDProperty " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetLEDProperty command sent successfully.")
    else:
        print("Failed to send SetLEDProperty command.")

def handle_get_network_property(send_mode, target_ip):
    """P.25 Get Network Information GetNetworkProperty"""
    print("\n--- GetNetworkProperty ---")
    command = "GetNetworkProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get network property.")

def handle_set_network_property(send_mode, target_ip):
    """P.26 Set Network Information SetNetworkProperty
    *This process may take up to 20 seconds for a response."""
    print("\n--- SetNetworkProperty ---")
    dhcp = ""
    while dhcp not in ['false', 'true']:
        dhcp = input("DHCP (false/true, required): ")
        if dhcp not in ['false', 'true']:
            print("Invalid value for DHCP. Please enter 'false' or 'true'.")

    ip_address = ""
    subnet_mask = ""
    gateway = ""
    dns = ""

    params = []
    params.append(f"DHCP={dhcp}")

    if dhcp == 'false':
        while not ip_address:
            ip_address = input("IPAddress (required when DHCP is 'false'): ")
            if not ip_address: print("IPAddress is required.")
        while not subnet_mask:
            subnet_mask = input("SubnetMask (required when DHCP is 'false'): ")
            if not subnet_mask: print("SubnetMask is required.")
        while not gateway:
            gateway = input("Gateway (required when DHCP is 'false'): ")
            if not gateway: print("Gateway is required.")
        while not dns:
            dns = input("DNS (required when DHCP is 'false'): ")
            if not dns: print("DNS is required.")

        params.append(f"IPAddress={ip_address}")
        params.append(f"SubnetMask={subnet_mask}")
        params.append(f"Gateway={gateway}")
        params.append(f"DNS={dns}")
    elif ip_address or subnet_mask or gateway or dns:
        print("Warning: IPAddress, SubnetMask, Gateway, DNS will be ignored when DHCP is 'true'.")
```

```
    command = "SetNetworkProperty " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip, timeout_seconds=20)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetNetworkProperty command sent successfully.")
        print("NOTE: If IP changed, response might not be received.")
    else:
        print("Failed to send SetNetworkProperty command.")

def handle_get_datetime_property(send_mode, target_ip):
    """P.27 Get Date/Time Information GetDatetimeProperty"""
    print("\n--- GetDatetimeProperty ---")
    command = "GetDatetimeProperty"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get datetime property.")

def handle_set_time_zone(send_mode, target_ip):
    """P.28 Set Time Zone SetTimeZone"""
    print("\n--- SetTimeZone ---")
    timezone_location = ""
    while not timezone_location:
        timezone_location = input("TimezoneLocation (e.g., Asia, required): ")
        if not timezone_location:
            print("TimezoneLocation is required.")

    timezone_city = ""
    while not timezone_city:
        timezone_city = input("TimezoneCity (e.g., Tokyo, required): ")
        if not timezone_city:
            print("TimezoneCity is required.")

    params = []
    params.append(f"TimezoneLocation={timezone_location}")
    params.append(f"TimezoneCity={timezone_city}")

    command = "SetTimeZone " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetTimeZone command sent successfully.")
    else:
        print("Failed to send SetTimeZone command.")

def handle_set_datetime_property(send_mode, target_ip):
    """P.29 Set Date/Time Information SetDatetimeProperty"""
    print("\n--- SetDatetimeProperty ---")
    sync_protocol = ""
    while sync_protocol not in ['0', '1']:
        sync_protocol = input("SyncProtocol (0:NTP, 1:SoftwareSync, required): ")
        if sync_protocol not in ['0', '1']:
            print("Invalid value for SyncProtocol. Please enter 0 or 1.")

    params = []
    params.append(f"SyncProtocol={sync_protocol}")

    if sync_protocol == '1': # For SoftwareSync
        domain_no = ""
```

```
while not domain_no.isdigit() or not (0 <= int(domain_no) <= 127): # DomainNo is required
    domain_no = input("DomainNo (0-127, required, SoftwareSync only): ")
    if not domain_no.isdigit() or not (0 <= int(domain_no) <= 127):
        print("Invalid value for DomainNo. Please enter a number between 0 and 127.")
params.append(f"DomainNo={domain_no}")

order_ps = ""
while order_ps not in ['0', '1']: # OrderPs is required
    order_ps = input("OrderPs (0:Primary, 1:Secondary, required, SoftwareSync only): ")
    if order_ps not in ['0', '1']:
        print("Invalid value for OrderPs. Please enter 0 or 1.")
params.append(f"OrderPs={order_ps}")
```

```
elif sync_protocol == '0': # For NTP
    ntp_synchronization = ""
    while ntp_synchronization not in ['true', 'false']:
        ntp_synchronization = input("NTPSynchronization (true/false, required, NTP only): ")
        if ntp_synchronization not in ['true', 'false']:
            print("Invalid value for NTPSynchronization. Please enter true or false.")
    params.append(f"NTPSynchronization={ntp_synchronization}")

    if ntp_synchronization == 'true':
        ntp_server = ""
        while not ntp_server:
            ntp_server = input("NTPServer (required when NTP synchronization is 'true'): ")
            if not ntp_server: print("NTPServer is required.")
        params.append(f"NTPServer={ntp_server}")

    date_str = input("Date (YYYY/MM/DD, leave blank for no change, ignored if NTP sync is OFF): ")
    time_str = input("Time (hh:mm:ss, leave blank for no change, ignored if NTP sync is OFF): ")
    if date_str: params.append(f"Date={date_str}")
    if time_str: params.append(f"Time={time_str}")

    command = "SetDatetimeProperty " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetDatetimeProperty command sent successfully.")
    else:
        print("Failed to send SetDatetimeProperty command.")

def handle_set_account_property(send_mode, target_ip):
    """P.30 Change Web Account Information SetAccountProperty"""
    print("\n--- SetAccountProperty ---")
    current_username = ""
    while not current_username:
        current_username = input("CurrentUserName (required): ")
        if not current_username:
            print("CurrentUserName is required.")

    current_password = ""
    while not current_password:
        current_password = input("CurrentPassword (required): ")
        if not current_password:
            print("CurrentPassword is required.")

    new_username = input("UserName (new, optional): ")
    new_password = input("Password (new, optional): ")

    params = [f"CurrentUserName={current_username}", f"CurrentPassword={current_password}"]
    if new_username: params.append(f"UserName={new_username}")
    if new_password: params.append(f"Password={new_password}")

    command = "SetAccountProperty " + "&".join(params)
```

```
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetAccountProperty command sent successfully.")
    else:
        print("Failed to send SetAccountProperty command.")

def handle_exec_initialize(send_mode, target_ip):
    """P.31 System Initialization ExecInitialize
    *This process may take up to 20 seconds for a response. The device will automatically restart after this command."""
    print("\n--- ExecInitialize ---")
    all_init = ""
    while all_init not in ['0', '1']:
        all_init = input("All (0:Full Initialization, 1:Partial Initialization, required): ")
        if all_init not in ['0', '1']:
            print("Invalid value for All. Please enter 0 or 1.")

    params = [f"All={all_init}"]

    if all_init == '1': # For partial initialization, at least one item must be true
        player = input("Player (true/false, optional): ")
        geometry = input("Geometry (true/false, optional): ")
        network = input("Network (true/false, optional): ")
        content = input("Content (true/false, optional): ")

        if player: params.append(f"Player={player}")
        if geometry: params.append(f"Geometry={geometry}")
        if network: params.append(f"Network={network}")
        if content: params.append(f"Content={content}")

        # For partial initialization, at least one item must be true
        if not (player == 'true' or geometry == 'true' or network == 'true' or content == 'true'):
            print("For partial initialization, at least one item (Player, Geometry, Network, Content) must be true. Exiting.")
            return

    command = "ExecInitialize " + "&".join(params)
    print("WARNING: This command will restart the FMP device. Proceed with caution.")
    confirm = input("Are you sure you want to initialize? (y/n): ").lower()
    if confirm == 'y':
        responses = execute_command(send_mode, command, target_ip, timeout_seconds=20)
        if responses and all(r.startswith("OK") for r in responses.values()):
            print("ExecInitialize command sent successfully. FMP will restart.")
        else:
            print("Failed to send ExecInitialize command.")
    else:
        print("Initialization cancelled.")


def handle_reboot(send_mode, target_ip):
    """P.32 System Reboot Reboot"""
    print("\n--- Reboot ---")
    command = "Reboot"
    print("WARNING: This command will restart the FMP device.")
    confirm = input("Are you sure you want to reboot? (y/n): ").lower()
    if confirm == 'y':
        responses = execute_command(send_mode, command, target_ip)
        if responses and all(r.startswith("OK") for r in responses.values()):
            print("Reboot command sent successfully. FMP will restart.")
        else:
            print("Failed to send Reboot command (this might be expected if FMP reboots quickly).")
    else:
        print("Reboot cancelled.")

def handle_get_log_level(send_mode, target_ip):
```

```
    """P.33 Get Log Output Level GetLogLevel"""
    print("\n--- GetLogLevel ---")
    command = "GetLogLevel"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    print(f"  {key}: {value}")
    else:
        print("Failed to get log level.")

def handle_set_log_level(send_mode, target_ip):
    """P.34 Set Log Output Level SetLogLevel"""
    print("\n--- SetLogLevel ---")
    log_level = ""
    while log_level not in ['0', '1', '2', '3']:
        log_level = input("LogLevel (0:DEBUG, 1:INFO, 2:WARNING, 3:FATAL, required): ")
        if log_level not in ['0', '1', '2', '3']:
            print("Invalid value for LogLevel. Please enter 0, 1, 2, or 3.")

    params = []
    params.append(f"LogLevel={log_level}")

    command = "SetLogLevel " + "&".join(params)
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("SetLogLevel command sent successfully.")
    else:
        print("Failed to send SetLogLevel command.")

# --- Main Interaction Loop ---
if __name__ == "__main__":
    current_target_ip = DEFAULT_FMP_IP  # Current target IP
    current_send_mode = '1'  # Default is single FMP ('1': Single FMP, '2': Broadcast)

    while True:
        print("\n--- FMP Configuration Control Tool ---")
        print(f"Current Target: {'Single FMP' if current_send_mode == '1' else 'Broadcast'} "
              f"{'(' + current_target_ip + ')' if current_send_mode == '1' else ''}")
        print("-------------------------------------")
        print("Configuration Commands:")
        print("  1: GetPlayerState")
        print("  2: GetPlayerProperty")
        print("  3: SetPlayerProperty (Max 30s response, REQUIRED fields enforced)")
        print("  4: GetAudioProperty")
        print("  5: SetAudioProperty (REQUIRED fields enforced)")
        print("  6: GetAdvancedProperty")
        print("  7: SetAdvancedProperty (REQUIRED fields enforced)")
        print("  8: GetStreamList")
        print("  9: PlayStream")
        print("  a: StopStream")
        print("  b: GetAutoPlayProperty")
        print("  c: SetAutoPlayProperty (REQUIRED fields enforced)")
        print("  d: GetPlaylistSyncProperty")
        print("  e: SetPlaylistSyncProperty (REQUIRED fields enforced)")
        print("  f: GetLanguage")
        print("  g: SetLanguage (REQUIRED fields enforced)")
        print("  h: GetHostName")
        print("  i: SetHostName (REQUIRED fields enforced)")
        print("  j: GetLEDProperty")
```

```
print("  k: SetLEDProperty (REQUIRED fields enforced)")
print("  l: GetNetworkProperty")
print("  m: SetNetworkProperty (Max 20s response, REQUIRED fields enforced)")
print("  n: GetDatetimeProperty")
print("  o: SetTimeZone (REQUIRED fields enforced)")
print("  p: SetDatetimeProperty (REQUIRED fields enforced)")
print("  q: SetAccountProperty (REQUIRED fields enforced)")
print("  r: ExecInitialize (WARNING: FMP will restart, Max 20s response, REQUIRED fields enforced)")
print("  s: Reboot (WARNING: FMP will restart)")
print("  t: GetLogLevel")
print("  u: SetLogLevel (REQUIRED fields enforced)")
print("\nUtility:")
print("  z: Change Target (Single FMP / Broadcast)")
print("  0: Exit")
print("-------------------------------------")

choice = input("Enter command number/letter: ").lower()

if choice == '1':
    handle_get_player_state(current_send_mode, current_target_ip)
elif choice == '2':
    handle_get_player_property(current_send_mode, current_target_ip)
elif choice == '3':
    handle_set_player_property(current_send_mode, current_target_ip)
elif choice == '4':
    handle_get_audio_property(current_send_mode, current_target_ip)
elif choice == '5':
    handle_set_audio_property(current_send_mode, current_target_ip)
elif choice == '6':
    handle_get_advanced_property(current_send_mode, current_target_ip)
elif choice == '7':
    handle_set_advanced_property(current_send_mode, current_target_ip)
elif choice == '8':
    handle_get_stream_list(current_send_mode, current_target_ip)
elif choice == '9':
    handle_play_stream(current_send_mode, current_target_ip)
elif choice == 'a':
    handle_stop_stream(current_send_mode, current_target_ip)
elif choice == 'b':
    handle_get_autoplay_property(current_send_mode, current_target_ip)
elif choice == 'c':
    handle_set_autoplay_property(current_send_mode, current_target_ip)
elif choice == 'd':
    handle_get_playlist_sync_property(current_send_mode, current_target_ip)
elif choice == 'e':
    handle_set_playlist_sync_property(current_send_mode, current_target_ip)
elif choice == 'f':
    handle_get_language(current_send_mode, current_target_ip)
elif choice == 'g':
    handle_set_language(current_send_mode, current_target_ip)
elif choice == 'h':
    handle_get_host_name(current_send_mode, current_target_ip)
elif choice == 'i':
    handle_set_host_name(current_send_mode, current_target_ip)
elif choice == 'j':
    handle_get_led_property(current_send_mode, current_target_ip)
elif choice == 'k':
    handle_set_led_property(current_send_mode, current_target_ip)
elif choice == 'l':
    handle_get_network_property(current_send_mode, current_target_ip)
elif choice == 'm':
    handle_set_network_property(current_send_mode, current_target_ip)
elif choice == 'n':
```

```
            handle_get_datetime_property(current_send_mode, current_target_ip)
        elif choice == 'o':
            handle_set_time_zone(current_send_mode, current_target_ip)
        elif choice == 'p':
            handle_set_datetime_property(current_send_mode, current_target_ip)
        elif choice == 'q':
            handle_set_account_property(current_send_mode, current_target_ip)
        elif choice == 'r':
            handle_exec_initialize(current_send_mode, current_target_ip)
        elif choice == 's':
            handle_reboot(current_send_mode, current_target_ip)
        elif choice == 't':
            handle_get_log_level(current_send_mode, current_target_ip)
        elif choice == 'u':
            handle_set_log_level(current_send_mode, current_target_ip)
        elif choice == 'z':
            print("\n--- Change Target Mode ---")
            mode_choice = input("Select send mode (1: Single FMP, 2: Broadcast): ")
            if mode_choice == '1':
                current_send_mode = '1'
                current_target_ip = input(f"Enter target FMP IP address (default: {DEFAULT_FMP_IP}): ") or DEFAULT_FMP_IP
                print(f"Target set to Single FMP: {current_target_ip}")
            elif mode_choice == '2':
                current_send_mode = '2'
                current_target_ip = None  # Not applicable for broadcast in this context
                print(f"Target set to Broadcast (using {BROADCAST_IP})")
            else:
                print("Invalid mode selection. Keeping current mode.")
        elif choice == '0':
            print("Exiting program.")
            break
        else:
            print("Invalid command. Please enter again.")

    sock.close()
    print("Socket closed.")
```

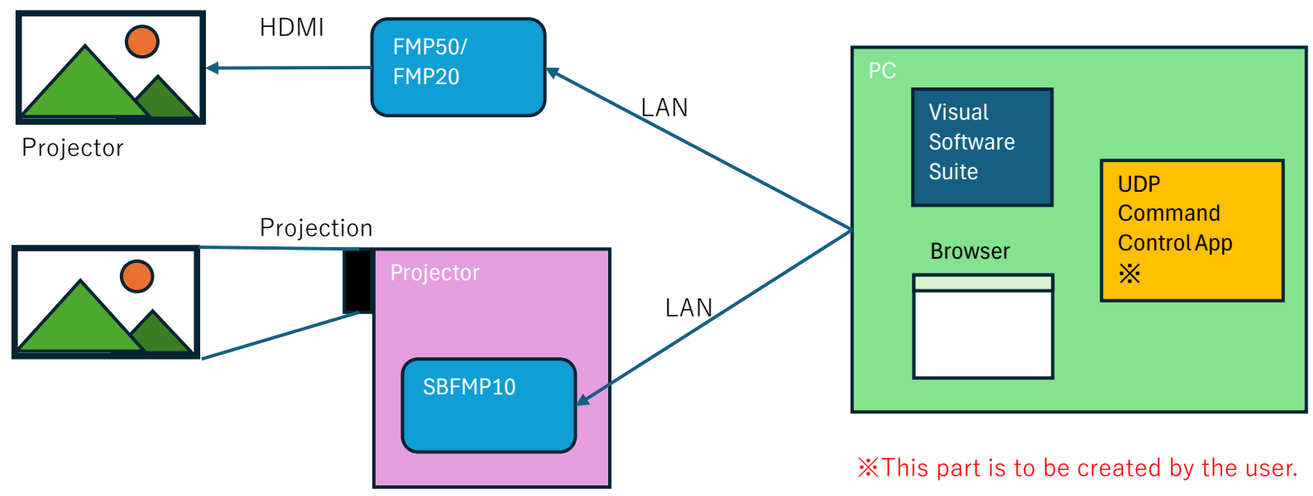
■How to Use UDP Commands (Playback Control)

● Overview

The Media Processor (FMP) allows playback operations of media using the Timeline function through UDP command communication.
Media playback operations are performed on a playlist (※) basis.
The Media Processor (FMP) can register up to 99 playlists, and media playback operations can be performed by switching between playlists.
Additionally, synchronized playback of media can be performed by multiple Media Processors (FMPs).

※：A playlist can contain multiple individual media (videos and still images) and perform playback operations as a single media.

● Configuration Diagram

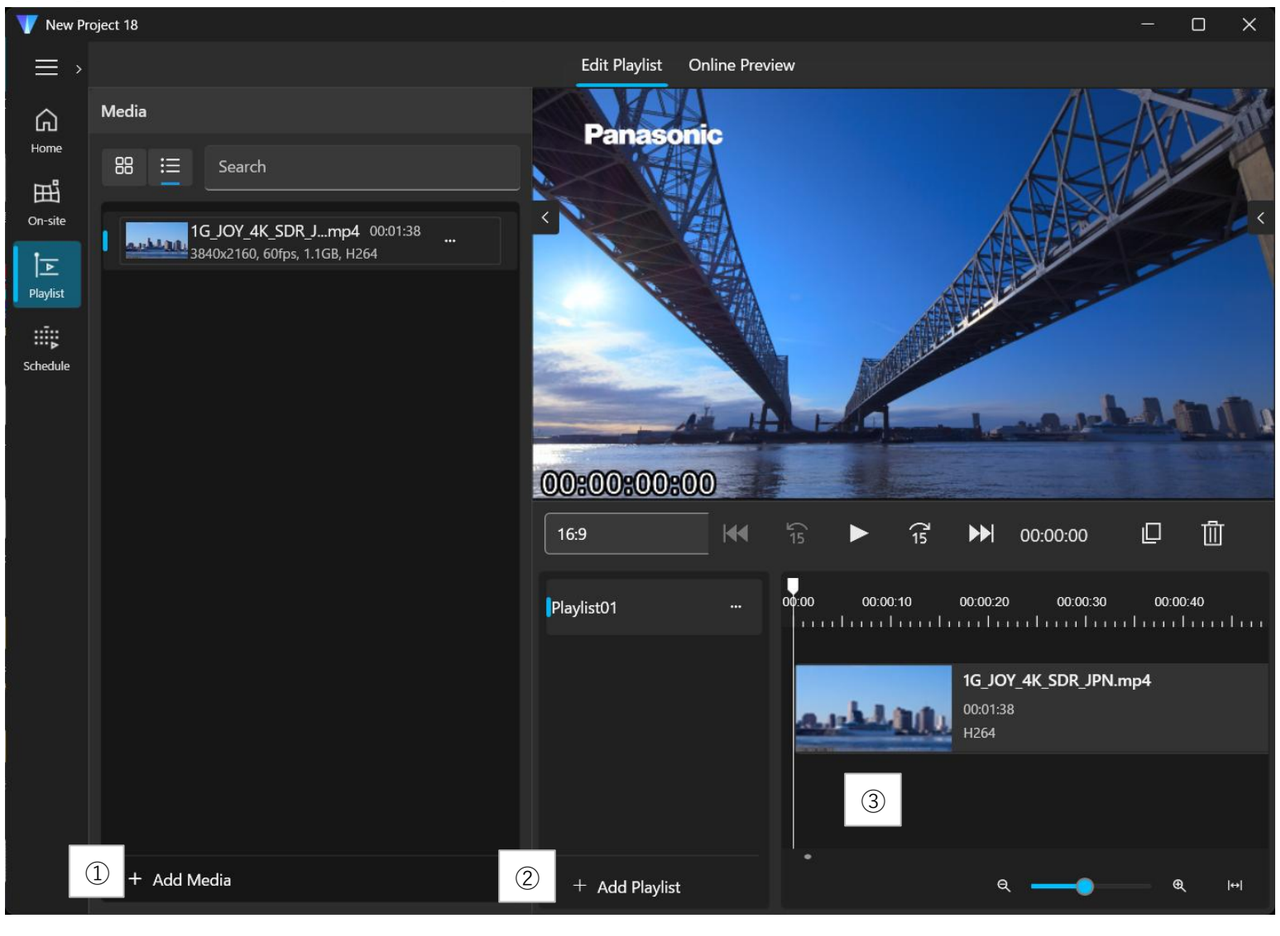


※This part is to be created by the user.

● Creating a playlist and sending it to the Media Processor using VSS

※：Please refer to the Visual Software Suite (VSS) manual for details.

1. Playlists are created on the playlist editing screen below.
   ① Add media to be registered in the playlist.
   ② Create a playlist.
   ③ Register the media added in ① to the playlist created in ②.



2. The created playlist is sent to the Media Processor (FMP) via the online preview screen.

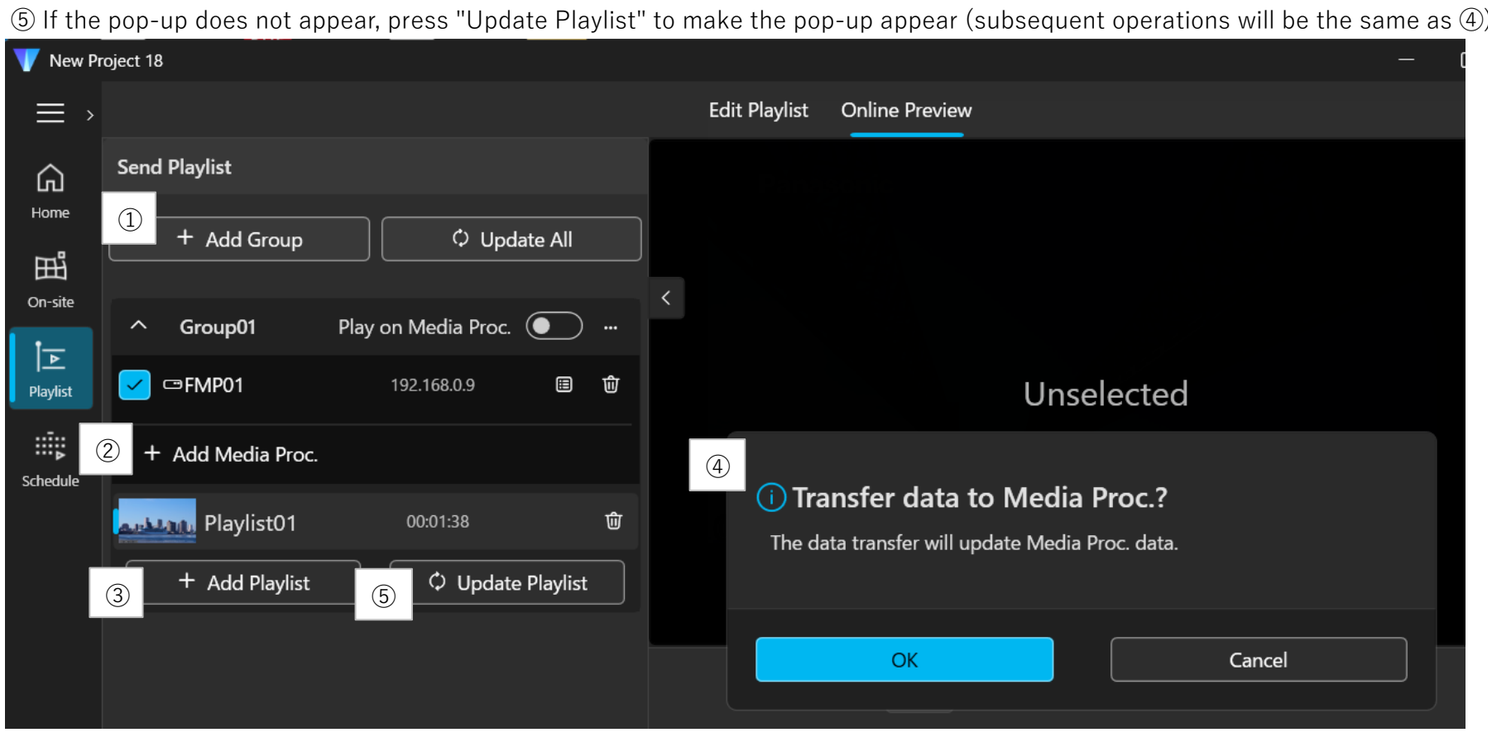
① Add a group to send the playlist to.

② Add the Media Processor (FMP) to send the playlist to.

③ Add the playlist created in the playlist editor.

④ After performing ③, a data transfer pop-up will appear. Press OK to send the playlist to the Media Processor (FMP).

⑤ If the pop-up does not appear, press "Update Playlist" to make the pop-up appear (subsequent operations will be the same as ④).



● **Procedures for launching Player: Timeline via WEB (FMP Series Control Window)**

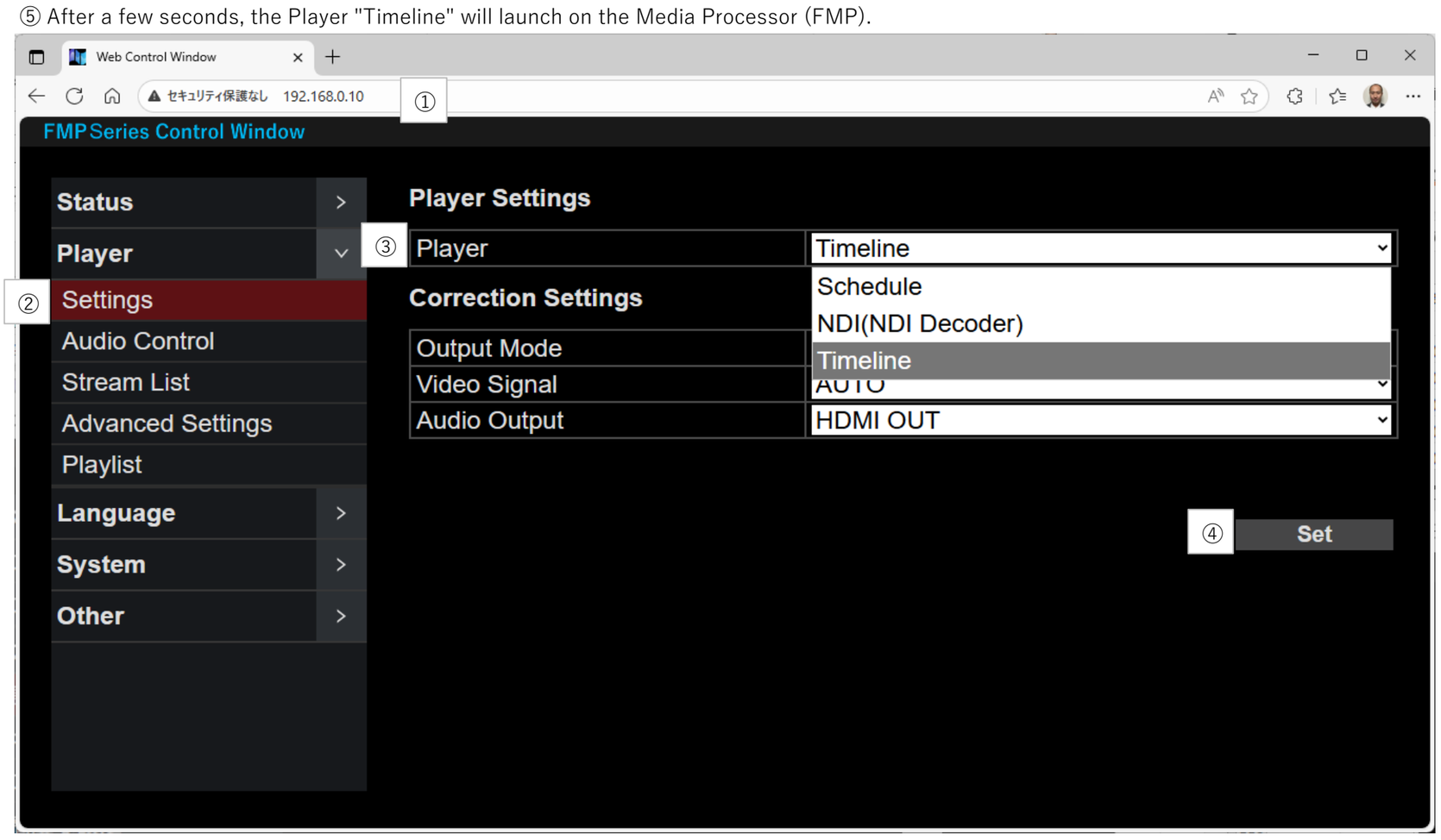
① Access the IP address of the corresponding Media Processor (FMP) in a web browser.

② From the left menu, select "Player" → "Settings" to transition to the screen below.

③ Select "Timeline" from "Player" in the Player Settings.

④ Press the "Set" button.

⑤ After a few seconds, the Player "Timeline" will launch on the Media Processor (FMP).

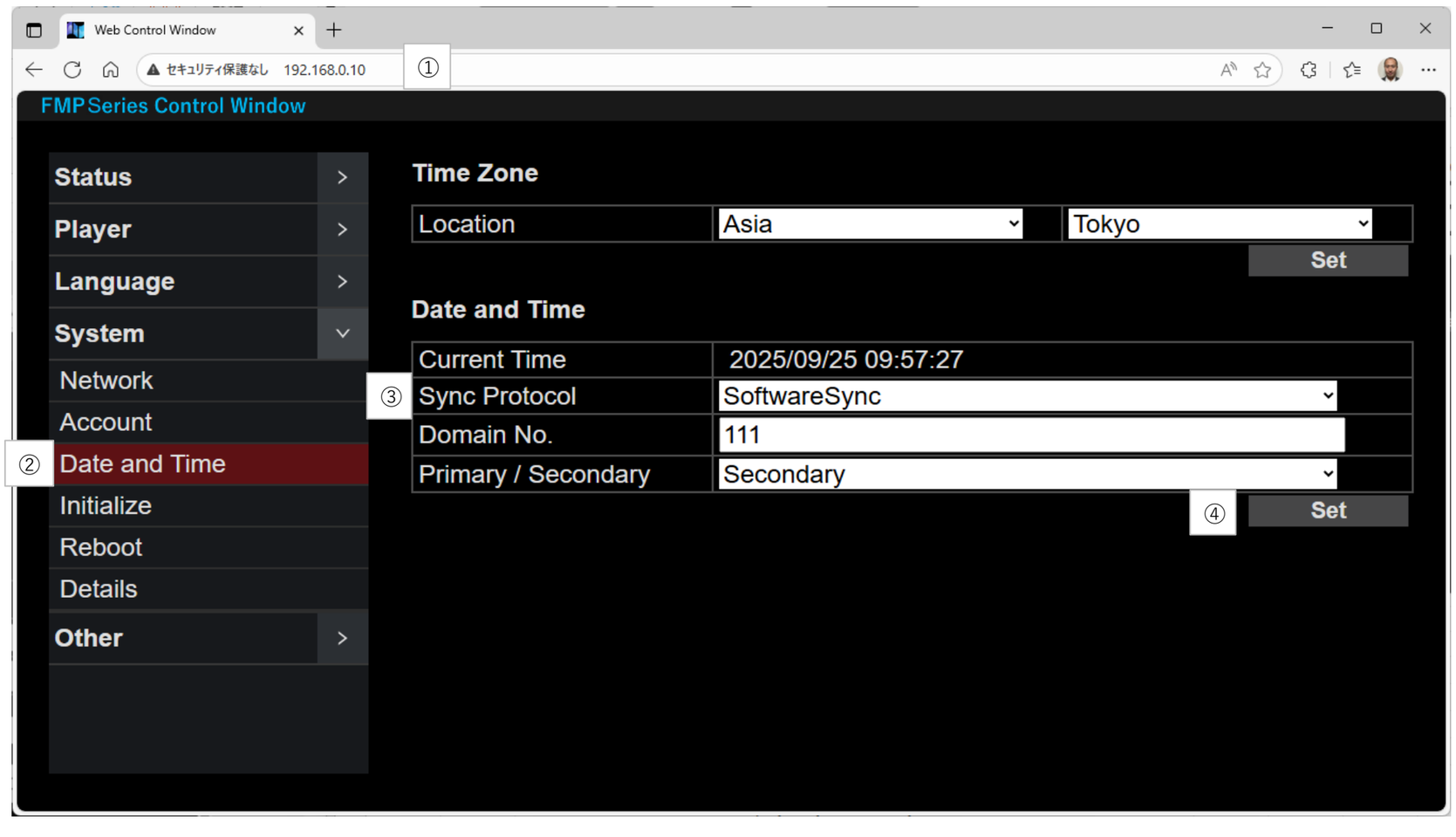


● Procedures for performing media synchronized playback via WEB (FMP Series Control Window)

※：This section describes the settings for media synchronized playback using two Media Processors. Please refer to the manual for details.

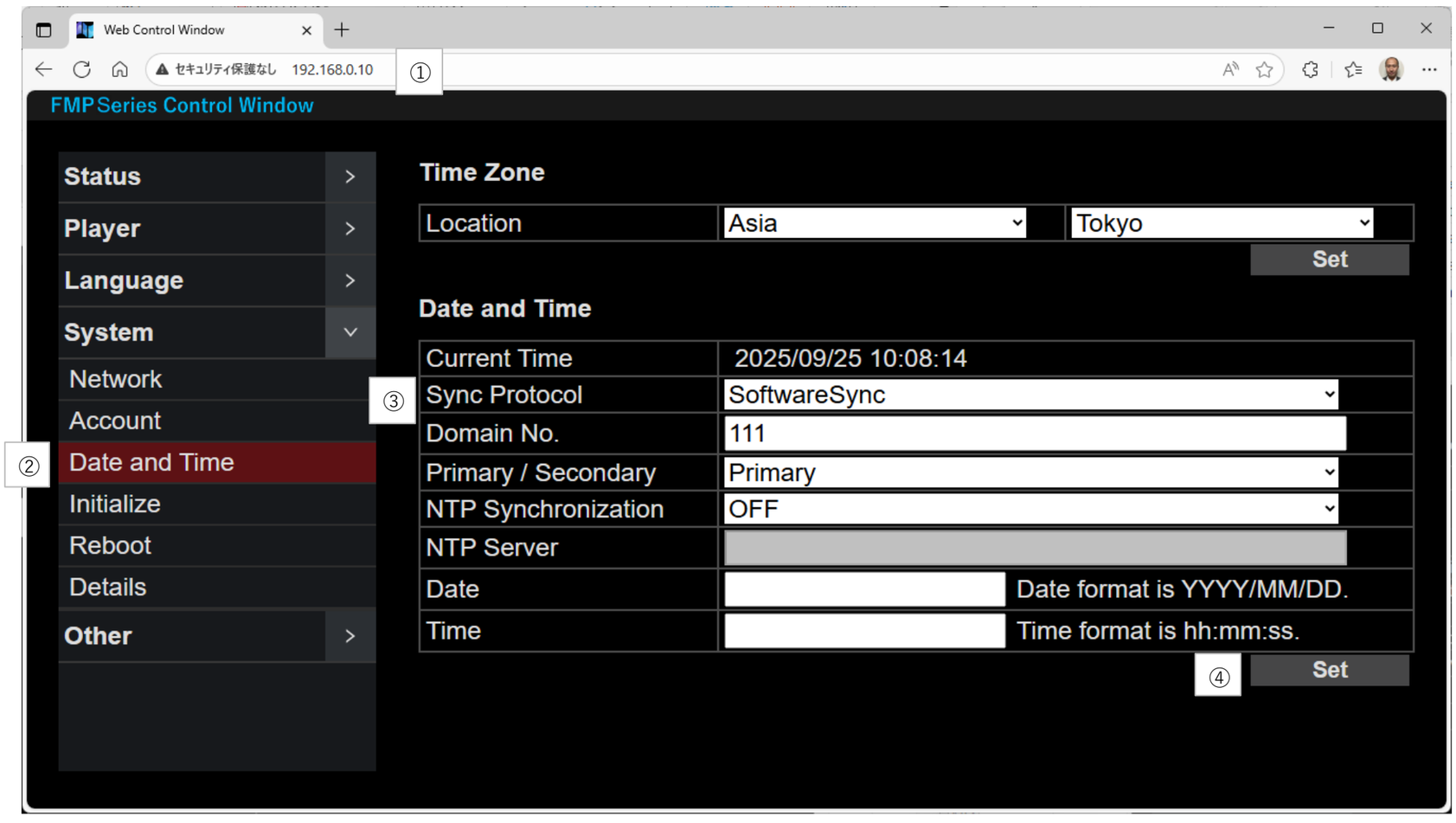
・Secondary side

① Access the IP address of the Media Processor (FMP) to be set as secondary in a web browser.
② From the left menu, select "System" → "Date and Time" to transition to the screen below.
③ From "Date and Time", select "SoftwareSync" for "Synchronization Protocol",
set an arbitrary value (here, "111") for "Domain Number",
and select "Secondary" from "Primary/Secondary".
④ Press the "Set" button.



・Primary side

① Access the IP address of the Media Processor (FMP) to be set as primary in a web browser.
② From the left menu, select "System" → "Date and Time" to transition to the screen below.
③ From "Date and Time", select "SoftwareSync" for "Synchronization Protocol",
set an arbitrary value (here, "111") for "Domain Number",
and select "Primary" from "Primary/Secondary".
④ Press the "Set" button.



● UDP Command Specifications

FMP Connection Conditions

Player: Timeline is running and waiting on the following socket.

IP address: IP address set in FMP

Listening port number: **65432**

Source address: **INADDR_ANY**

Maximum transmit and receive size: **65536byte**

Operation Command List

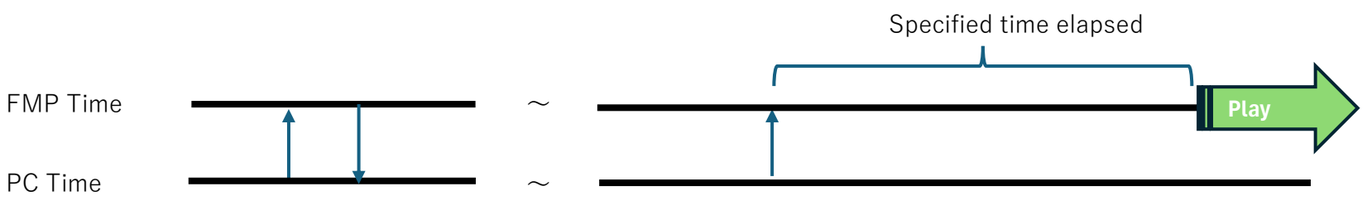
Command	説明
GetSystemTime	**Time adjustment between UDP command execution side and FMP side**
Play	**Playlist playback**
Pause	**Stop playlist playback** Stop playback at the specified position
Stop	**Stop playlist playback** Playback stops when the command is received (You do not need to specify the stop position.)
Seek	**Seeking playlist playback**
Skip	**Skipping media in playlists**
SelectPlayList	**Playlist Selection**
Loop	**Playlist loop settings**
DisconnectReq	**Restarting the Timeline Module**
ResetPlaylist	**Reload a playlist**
SetSyncTime	**FMP Time adjustment setting** (initial value is 3 seconds)

Information Command List

Command	説明
GetPlaylistInfo	**Retrieving Playlist Information**
GetEachPlaylistInfo	**Retrieving media information in a playlist**
GetPlayingInfo	**Retrieving playlist playback status**

Procedure for playlist playback (when specifying synchronized start time)

【Prosessing Image】



GetSystemTime (Command)
(requesting time difference of internal clock)

Play (Command)
Specify synchronized start time in SyncTime
(Example: Specified time = current time + internal clock time difference + N seconds)

[Steps]

① Execute GetSystemTime (time adjustment between UDP command sender and FMP).
② Calculate the time difference between the internal clocks of the UDP command sender PC and the FMP,
and add this value to the "synchronized start time" parameter of subsequent UDP commands.
③ Execute SelectPlaylist (select playlist).
④ Execute Loop (set playlist loop).
⑤ Execute Play (playlist playback).

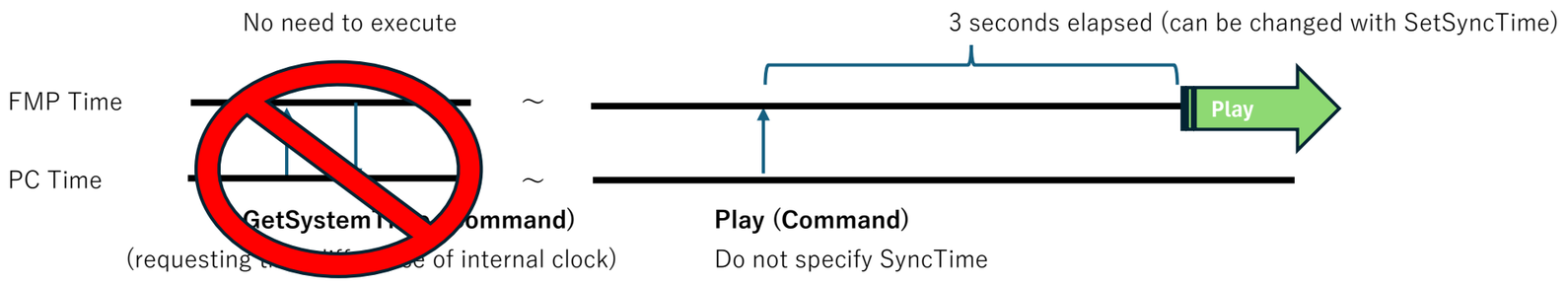
Thereafter, use Pause, Stop, Seek, and Skip commands to play/stop playback of the playlist.

[Command Examples]

① GetSystemTime CurrentTimeVss=1715237197598000
② -
③ SelectPlaylist PlayListName=Playlist01
④ Loop EnableLoop=0
⑤ Play SyncTime=1727684152203481&PlayPosition=0

Procedure for playlist playback (when not specifying synchronized start time)

【Prosessing Image】



[Steps]

① Execute SelectPlaylist (select playlist).
② Execute Loop (set playlist loop).
③ Execute Play (playlist playback). ※

Thereafter, use Stop, Seek※, and Skip※ commands to play/stop playback of the playlist.

※：FMP performs playback etc. 3 seconds after receiving the command (can be changed with SetSyncTime command)

[Command Examples]

① SelectPlayList PlayListName=Playlist01
② Loop EnableLoop=0
③ Play

Checking playlist playback status

[Steps]

① Set playlist to playback status.
② Execute GetPlayingInfo (get playlist playback status) at any timing.
③ Determine playlist playback status by the increase in CurrentPosition (current position of playlist) from the return of GetPlayingInfo (get playlist playback status).

Reloading playlist after update

[Steps]

① Execute playlist update from VSS to FMP.
② Execute DisconnectReq (module reboot) or ResetPlaylist (reload playlist).
③ Execute GetPlaylistInfo (get playlist information) and GetEachPlaylistInfo (get media information in playlist) to confirm that the playlist has been updated.
④ Execute "Procedure for playlist playback".

	Time adjustment between UDP command execution side and FMP side

Command	**GetSystemTime**

Description

To determine the time difference between the internal clock on the PC executing the UDP command and the FMP side.

The time difference should be calculated in microseconds, and this value should be added to the parameter "playback start synchronization time" of the subsequent UDP command.

Request

Parameter	Always	Description	Remarks
CurrentTimeVss	○	UDP Commands execution time (LinuxTime + usec)	Linux time format (including usec after the decimal point) multiplied by 1000000

Request Example

GetSystemTime CurrentTimeVss=1715237197598000

*1: The separator between UDP commands and parameters is " " (space), and the value of the parameter is after "=".
*2: 1715237197598000＝2024/05/09 06:46:37.598000

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
CurrentTimeVss	○	Execution time set at the time of request	
CurrentTimeFmp	○	FMP Time(Linux time + usec)	Linux time format (including usec after the decimal point) multiplied by 1000000

Return Example

On Success

OK CurrentTimeVss=1715237197598000&CurrentTimeFmp=1715237196004200

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
※２：1715237197598000＝2024/05/09 06:46:37.598000
　　1715237196004200＝2024/05/09 06:46:36.004200

On Failure

None

	Playlist playback

Command	**Play**

Description

To play the specified playlist registered in FMP.

Request

Parameter	Always	Description	Remarks
SyncTime		Regeneration start time (LinuxTime + usec)	Linux time format (including usec after the decimal point) multiplied by 1000000 Specify a time that is at least 1 seconds ahead of the current time Also, include the time difference in GetSystemTime
PlayPosition		Playlist playback position (elapsed time from the beginning)	Linux time format (including usec after the decimal point) multiplied by 1000000

Request Example

When specifying synchronized playback start time: Play SyncTime=1727684152203481&PlayPosition=0
When not specifying synchronized playback start time: Play PlayPosition=0 or Play

*1: The separator between the execution result and the parameter is " " (space), the delimiter between the parameters is "&", and the value of the parameter is after "=".
*2: If the playback position of the playlist is set to something other than 0, playback can be started from the middle of the playlist.
*3:If synchronized playback start time is not specified, the "SyncTime" and "PlayPosition" parameters can be omitted. In that case, the FMP will start playback from the beginning of the playlist 3 seconds after receiving this command (changeable with SetSyncTime).

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	
NG Parameter		Cause of NG	

Return Example

On Success

OK

On Failure

NG

*: NG may be due to the following factors.
- Playlist is not specified.
- The playback start synchronization time points to the past.
- The playback position of the playlist points outside the range of the playlist.

	Stop playlist playback

Command	**Pause**

Description

To stop playback of a playlist at the specified stop position of the playlist.
Stop while displaying the video of the stop position.

Request

Parameter	Always	Description	Remarks
StopPosition	○	Playlist stop position (elapsed time from the beginning)	Linux time format (including usec after the decimal point) multiplied by 1000000

Request Example

Pause StopPosition=3800490

※：The separator between UDP commands and parameters is " " (space), and the value of the parameter is after "=".

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

On Failure

NG

*: NG may be due to the following factors.
- The stop position of the playlist points outside the range of the playlist.

	Stop playlist playback

Command	Stop

Description

To stop playback of the playlist at the position at the time of receiving the command.
The video at the time of receiving the command is displayed and stopped.

Request

Parameter	Always	Description	Remarks
None			

Request Example

Stop

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	
CurrentPosition	○	Playlist stop position (elapsed time from the beginning)	Linux time format (including usec after the decimal point) multiplied by 1000000 If you set this value to the playback position of playlist playback, the playlist will be played from the stopped position.

Return Example

On Success

OK CurrentPosition=10000000

*1: The separator between the execution result and the parameter is " " (space), and the value of the parameter is after "=".
*2: 10000000 microseconds = 10 seconds

On Failure

NG CurrentPosition=-1

*: NG may be due to the following factors.
- Playlist is not playing.
- Playback stop in FMP is in progress.

	Seeking playlist playback

Command	**Seek**

Description

To play a playlist from the playlist playback position specified during playlist playback.

If playlist playback is stopped, it will only move to the playback position of the specified playlist.

Request

Parameter	Always	Description	Remarks
SyncTime		Regeneration start time (LinuxTime + usec)	Linux time format (including usec after the decimal point) multiplied by 1000000 Specify a time that is at least 1 seconds ahead of the current time Also, include the time difference in GetSystemTime
PlayPosition	○	Playlist playback position (elapsed time from the beginning)	Linux time format (including usec after the decimal point) multiplied by 1000000

Request Example

When specifying synchronized playback start time: Seek SyncTime=1727669330084099&PlayPosition=0
When not specifying synchronized playback start time: Seek PlayPosition=0

*1: The separator between UDP commands and parameters is " " (space), the separator between parameters is "&", and the parameter value is after "=".
*2: If the playback position of the playlist is set to something other than 0, it can be played or moved from the middle of the playlist.
*3: If synchronized playback start time is not specified, the "SyncTime" parameter can be omitted. In that case, the FMP will start playback 3 seconds after receiving this command (changeable with SetSyncTime).

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	
NG Parameter		Cause of NG	

Return Example

On Success

OK

On Failure

NG

*: NG may be due to the following factors.
- Playlist not specified -> Playlist not selected
- Playlist playback position is outside the playlist range -> Out of range PlayPosition: (Specified playback position)
- Another command is in progress -> Other command running

	Skipping media in playlists

Command	Skip

Description

to skip from the current media to the next in a playlist.

If the playlist playback is stopped, skipping will not be performed.
Also, if there is only one piece of media registered in the playlist, it will not be skipped and will stop playing.
(When the loop setting is turned on, it returns to the beginning.)

Request

Parameter	Always	Description	Remarks
None			

Request Example

Skip

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	
NG Parameter		Cause of NG	

Return Example

On Success

OK

On Failure

NG

*: NG may be due to the following factors.
- Playlist not specified -> Playlist not selected
- Playlist is not playing -> Playlist not playing
- Another command is in progress -> Other command running

	Playlist Selection

Command	**SelectPlayList**

Description

To select a playlist to play in a playlist registered in FMP.

Request

Parameter	Always	Description	Remarks
PlayListName	○	Playlist name to select	

Request Example

```
SelectPlayList PlayListName=Playlist01

*: The separator between UDP commands and parameters is " " (space), and the value of the parameter is after "=".
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

```
OK
```

On Failure

```
NG

*: NG may be due to the following factors.
  - The playlist does not exist in FMP.
```

	Playlist loop settings

Command	**Loop**

Description

To choose whether to loop the playlist playback within the playlist.

The initial value is no loop (stop at end).

Request

Parameter	Always	Description	Remarks
EnableLoop	○	Loop setting (0: no loop, 1: with loop)	No loop(stop at the end),loop(play again from the beginning)

Request Example

Loop EnableLoop=0

*: The separator between UDP commands and parameters is " " (space), and the value of the parameter is after "=".

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd

Return Example

On Success

OK

On Failure

None

	Restarting the Module

Command	**DisconnectReq**

Description

To restart the Timeline modules running in the FMP.

* Please restart the Timeline module when you replace the playlist.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
DisconnectReq
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd

Return Example

On Success

```
OK
```

On Failure

None

	Reload a playlist

Command	ResetPlaylist

Description

To update the playlist information imported into the module running in the FMP.

* Please use this command when "Replace Playlist".
  Further, when this Command is executed during playlist playback, playlist playback is stopped. After that, you need to select a playlist.

Request

Parameter	Always	Description	Remarks
None			

Request Example

ResetPlaylist

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd

Return Example

On Success

OK

On Failure

None

	FMP Time Adjustment Setting

Command	**SetSyncTime**

Description

Command to change the time adjustment value for command execution within the FMP when synchronized playback start time is not specified.
*The initial value is set to 3 seconds.
Also, this adjustment value will revert to 3 seconds when the FMP is restarted or the Timeline module is restarted.

Request

Parameter	Always	Description	Remarks
Time	○	Time adjustment value	Specify an integer for the number of seconds. If less than 3 seconds, synchronized playback may not be possible.

Request Example

SetSyncTime Time=5

*: The separator between UDP commands and parameters is " " (space), and the value of the parameter is after "=".

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd

Return Example

On Success

OK

On Failure

None

	Retrieving Playlist Information

Command	**GetPlaylistInfo**

Description

To get a list of playlist names registered in FMP.

Request

Parameter	Always	Description	Remarks
-d		Debugging Option	Notify with the ID part used in Visual Software Suite (VSS)

Request Example

```
GetPlaylistInfo

GetPlaylistInfo -d

*: The separator between Command and parameter is " " (space).
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	
Playlist1		Playlist name	Name is not available if NG or unregistered
Playlist2		Playlist name	Name is not available if NG or unregistered
...			
PlaylistN		Playlist name	Name is not available if NG or unregistered

Return Example

On Success

```
OK Playlist1=Weekday&Playlist2=Weekend

*1: The separator between the execution result and the parameter is " " (space), the delimiter between the parameters is "&", and the value of the parameter is after "=".
*2: In the above example, only two playlists are registered, so there is only "Playlist2".

OK Playlist1=Weekday--(761e4d70-d6c7-4abd-aed0-889288240ef1).list&Playlist2=Weekend--(ea51550d-627f-48da-9776-f5808c0aac23).list

*3: When the debug option is specified, the ID part is added as described above.
```

On Failure

```
NG

*: NG may be due to the following factors.
  - None of the playlists are registered.
```

	Retrieving media information in a playlist

Command	**GetEachPlaylistInfo**

Description

To get media information (media name, playback time) in a playlist registered in FMP.

Request

Parameter	Always	Description	Remarks
Playlist	○	The name of the playlist for which you want to retrieve media information	You don't need the ID part used by VisualSoftwareSuite (VSS)

Request Example

GetEachPlaylistInfo Playlist=Weekday

*: The separator between Command and parameter is " " (space), and the value of the parameter is after "=".

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	
Playlist	○	Playlist name set at the time of Request	
Medianum	○	Number of media in the playlist	
Media1		First Media Name	
Duration1		Duration of the first media	The unit is "seconds"
Media2		Second media name	
Duration2		Duration of the second media	The unit is "seconds"
…		・・・	
MediaN		"N"th Media Name	
DurationN		Duration of the "N"th media	The unit is "seconds"

Return Example

On Success

OK Playlist=PLAYLIST1&Medianum=3&Media1=MEDIA1.mp4&Duration1=60&Media2=MEDIA2.mp4&Duration2=60&Media3=MEDIA3.jpg&Duration3=10

*1: The separator between the execution result and the parameter is " " (space), the delimiter between the parameters is "&", and the value of the parameter is after "=".
*2: In the above example, only three media are registered in the playlist, so there are only "Media3" and "Duration3".
*3: If a large number of media is registered in the playlist, the maximum transmission and reception size: 1024 bytes may be exceeded, but in that case, up to 1024 bytes will be notified.

On Failure

NG Playlist=PLAYLIST1&Medianum=0

*: NG may be due to the following factors.
- The playlist does not exist.

	Retrieving playlist playback status

Command	**GetPlayingInfo**

Description

To get the status of playlist playback in FMP (determined by the current position of the playlist).

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetPlayingInfo
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	
Playlist	○	Playlist name	
CurrentPosition	○	Playlist current location	When playback is stopped, "STOP" and other NGs are blank
Loop	○	Loop setting (0: no loop, 1: with loop)	No loop(stop at the end),loop(play again from the beginning)

Return Example

On Success

```
OK Playlist=Weekday&CurrentPosition=100000000&Loop=0

*1: The separator between the execution result and the parameter is " " (space), the delimiter between the parameters is "&", and the value of the parameter is after "=".

OK Playlist=PLAYLIST1&CurrentPosition=STOP&Loop=0

*2: When playlist playback is stopped (after Pause or Stop is executed), the current position of the playlist is notified by "STOP".
```

On Failure

```
NG Playlist=&CurrentPosition=&Loop=0

*: NG may be due to the following factors.
  - Playlist is not specified.
```

●Sample code for Playback Control（python）

This is a sample code for playback control.

Please change 'DEFAULT_FMP_IP' and 'BROADCAST_IP' according to the IP address of the FMP you are using (local IP).

```
import socket
import time
import re

# --- Constant Settings ---
M_SIZE = 1024  # Receive buffer size

# FMP address and target port number (default for single FMP)
# Please change according to your actual FMP address
DEFAULT_FMP_IP = '192.168.0.11'  # Default IP for single FMP
FMP_PORT = 65432

# Broadcast address (usually .255 for IPv4 /24 subnet)
# Adjust if your subnet mask is different
BROADCAST_IP = '192.168.0.255'  # Example broadcast IP for a /24 subnet

# --- UDP Socket Initialization ---
# This socket will be reused, so ensure it's not bound to a specific IP for broadcast if needed
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)  # Enable broadcast mode for this socket

# --- Global Variables for Playback State ---
diff_time_us = 0  # Time difference with FMP (microseconds)
marge_time_us = 3000000  # FMP processing time allowance (3 seconds = 3,000,000 microseconds) - 3 seconds or more recommended
PLAY_POSITION = 0  # Playback start position (in us)
PLAYLIST_NAME = "Playlist01"  # Default playlist name
CURRENT_LOOP_STATUS = '0'  # 0: Loop disable, 1: Loop enable

# --- Helper Functions for sending commands ---

def send_command_single_fmp(target_ip, command_str):
    """
    Sends the specified UDP command string to a single FMP and receives a response.
    Returns a dictionary with FMP IP as key and response string as value.
    """
    target_address = (target_ip, FMP_PORT)
    print(f"Sending to {target_address}: {command_str}")
    try:
        sock.settimeout(10)  # Ensure timeout is set for single FMP
        sock.sendto(command_str.encode('utf-8'), target_address)
        rx_message, addr = sock.recvfrom(M_SIZE)
        decoded_message = rx_message.decode(encoding='utf-8')
        print(f"Received from {addr}: {decoded_message}")
        return {addr[0]: decoded_message}  # Return as a dictionary for consistency
    except socket.timeout:
        print(f"Error: Socket timeout when sending to {target_address}. FMP might not be responding.")
        return {target_ip: "ERROR: TIMEOUT"}
    except Exception as e:
        print(f"Error sending/receiving data to {target_address}: {e}")
        return {target_ip: f"ERROR: EXCEPTION ({e})"}

def send_command_broadcast(command_str, timeout_seconds=3, expected_fmp_count=None):
    """
    Sends the specified UDP command string via broadcast and collects responses from multiple FMPs.
    Returns a dictionary with FMP IP as key and response string as value for each FMP.
    """
    broadcast_address = (BROADCAST_IP, FMP_PORT)
    print(f"Sending broadcast to {broadcast_address}: {command_str}")

    responses = {}
    sock.settimeout(0.1)  # Shorter timeout for collecting multiple responses
    start_time = time.time()

    try:
        sock.sendto(command_str.encode('utf-8'), broadcast_address)

        while time.time() - start_time < timeout_seconds:
            try:
                rx_message, addr = sock.recvfrom(M_SIZE)
                decoded_message = rx_message.decode(encoding='utf-8')
                print(f"Received from {addr}: {decoded_message}")
                responses[addr[0]] = decoded_message  # Store response with IP as key

                if expected_fmp_count is not None and len(responses) >= expected_fmp_count:
                    print(f"Collected {len(responses)} responses, which is the expected count.")
                    break

            except socket.timeout:
                # No more packets in the buffer for this short timeout
                pass
            except Exception as e:
                print(f"Error receiving data in broadcast loop: {e}")
                break
```

```
        if not responses:
            print("No FMP responses received within the timeout period.")

        return responses

    except Exception as e:
        print(f"Error sending broadcast or setting up socket: {e}")
        return {}
    finally:
        # Reset timeout to default for single FMP operations
        sock.settimeout(10)

def execute_command(send_mode, command_str, target_ip=None):
    """Wrapper function to execute command based on send_mode."""
    if send_mode == '1':  # Single FMP
        if target_ip is None:  # Should not happen if target_ip is always passed for single mode
            target_ip = input(f"Enter target FMP IP address (default: {DEFAULT_FMP_IP}): ") or DEFAULT_FMP_IP
        return send_command_single_fmp(target_ip, command_str)
    elif send_mode == '2':  # Broadcast
        broadcast_timeout = float(input("Enter broadcast response collection timeout in seconds (e.g., 3.0): ") or 3.0)
        expected_count_input = input("Enter expected number of FMPs (leave blank if unknown): ")
        expected_count = int(expected_count_input) if expected_count_input else None
        return send_command_broadcast(command_str, timeout_seconds=broadcast_timeout, expected_fmp_count=expected_count)
    else:
        print("Invalid send mode. Please choose 1 or 2.")
        return {}

def parse_response_params(response_str):
    """
    Parses FMP's OK response string into a dictionary of parameters.
    Example: "OK Player=1&ByVss=0&Content=abc.mp4" -> {"Player": "1", "ByVss": "0", "Content": "abc.mp4"}
    """
    params = {}
    if not response_str.startswith("OK"):
        return params

    # Remove "OK " prefix and split by "&"
    param_pairs = response_str[3:].split('&')
    for pair in param_pairs:
        if '=' in pair:
            key, value = pair.split('=', 1)
            params[key] = value
        else:
            params[pair] = True  # For parameters like "-d" (debug option) without value
    return params

# --- Command Specific Helper Functions ---

def update_diff_time(send_mode, target_ip):
    """
    Executes the GetSystemTime command and updates the time difference (diff_time_us) with the FMP.
    For broadcast, it uses the time from the first FMP to respond.
    """
    global diff_time_us
    start_u = int(time.time() * 1000000)
    command = f"GetSystemTime CurrentTimeVss={start_u}"

    responses = execute_command(send_mode, command, target_ip)

    if responses:
        first_ip = list(responses.keys())[0]
        response = responses[first_ip]

        if response.startswith("OK"):
            fmp_time_match = re.search(r'CurrentTimeFmp=(\d+)', response)
            if fmp_time_match:
                fmp_time = int(fmp_time_match.group(1))
                diff_time_us = start_u - fmp_time
                print(f"System time difference (PC_Vss - FMP_Fmp) from {first_ip}: {diff_time_us} us")
                return diff_time_us
            else:
                print(f"Error: CurrentTimeFmp not found in GetSystemTime response from {first_ip}.")
                return 0
        else:
            print(f"GetSystemTime command failed for {first_ip}: {response}")
            return 0
    print("No FMP responded to GetSystemTime or an error occurred.")
    return 0

# --- Main Command Handling Functions ---

def handle_get_system_time(send_mode, target_ip):
    global diff_time_us
    diff_time_us = update_diff_time(send_mode, target_ip)
    print(f"Current System Time Difference (diff_time_us): {diff_time_us} us")

def handle_select_playlist(send_mode, target_ip):
    global PLAYLIST_NAME
```

```
    input_playlist_name = input(f"Enter playlist name (default: {PLAYLIST_NAME}): ") or PLAYLIST_NAME
    command = f"SelectPlayList PlayListName={input_playlist_name}"
    responses = execute_command(send_mode, command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        PLAYLIST_NAME = input_playlist_name
        print(f"Playlist selected: {PLAYLIST_NAME}")
    else:
        print(f"Failed to select playlist: {input_playlist_name}")

def handle_loop(send_mode, target_ip):
    global CURRENT_LOOP_STATUS
    print("\n--- Loop Setting ---")
    print("1: Loop disable (0)")
    print("2: Loop enable (1)")
    loop_choice = input(f"Select loop status (current: {CURRENT_LOOP_STATUS}): ")
    if loop_choice == '1':
        CURRENT_LOOP_STATUS = '0'
    elif loop_choice == '2':
        CURRENT_LOOP_STATUS = '1'
    else:
        print("Invalid choice. Keeping current loop status.")
        return

    command = f"Loop EnableLoop={CURRENT_LOOP_STATUS}"
    responses = execute_command(send_mode, command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print(f"Loop status set to: {CURRENT_LOOP_STATUS}")
    else:
        print("Failed to set loop status.")

def handle_play(send_mode, target_ip):
    global PLAY_POSITION

    print("\n--- Play Command Settings ---")
    print("1: Play from start position (PlayPosition=0)")
    print("2: Play from current/stop position (uses PlayPosition from Stop/Pause)")
    play_pos_choice = input("Select play position: ")

    if play_pos_choice == '1':
        PLAY_POSITION = 0
    elif play_pos_choice == '2':
        # Assumes PLAY_POSITION has been updated during Stop/Pause
        print(f"Using current PLAY_POSITION: {PLAY_POSITION} us")
    else:
        print("Invalid choice. Exiting play settings.")
        return

    command_params = [f"PlayPosition={PLAY_POSITION}"]

    # Ask user whether to use SyncTime
    use_sync_time = input("Use SyncTime? (y/n): ").lower()
    if use_sync_time == 'y':
        if diff_time_us == 0:
            print("Warning: diff_time_us is 0. Running GetSystemTime first...")
            update_diff_time(send_mode, target_ip)

        # SyncTime = Current time (PC) + Marge time - Time difference
        sync_time_val = int(time.time() * 1000000) + marge_time_us - diff_time_us
        command_params.insert(0, f"SyncTime={sync_time_val}")  # Add SyncTime at the beginning

    # Construct the final command string
    # Play Command: "Play SyncTime=...&PlayPosition=..." or "Play PlayPosition=..."
    final_command = "Play " + "&".join(command_params)  # Command name and first parameter separated by space
    responses = execute_command(send_mode, final_command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print(f"Play command sent successfully for Playlist: {PLAYLIST_NAME}")
    else:
        print(f"Failed to send Play command for Playlist: {PLAYLIST_NAME}")

def handle_pause(send_mode, target_ip):
    global PLAY_POSITION

    current_pos_str = None
    print("Attempting to get current playing info for Pause command...")
    responses_info = execute_command(send_mode, "GetPlayingInfo", target_ip)

    if responses_info:
        first_ip = list(responses_info.keys())[0]
        response_str_info = responses_info[first_ip]
        if response_str_info.startswith("OK"):
            parsed_params = parse_response_params(response_str_info)
            current_pos_str = parsed_params.get('CurrentPosition')
        else:
            print(f"Failed to get playing info from {first_ip}: {response_str_info}")
            print("Cannot determine StopPosition. Using default pause position.")
            pause_position = 10000000  # Default value (10 seconds)
            command = f"Pause StopPosition={pause_position}"
            responses = execute_command(send_mode, command, target_ip)
            if responses and list(responses.values())[0].startswith("OK"):
```

```
            print(f"Pause command sent successfully with default StopPosition: {pause_position} us")
            PLAY_POSITION = pause_position
        else:
            print("Failed to send Pause command.")
        return

else:
    print("No FMP responded to GetPlayingInfo. Using default pause position.")
    pause_position = 10000000  # Default value (10 seconds)
    command = f"Pause StopPosition={pause_position}"
    responses = execute_command(send_mode, command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print(f"Pause command sent successfully with default StopPosition: {pause_position} us")
        PLAY_POSITION = pause_position
    else:
        print("Failed to send Pause command.")
    return

if current_pos_str:
    # Corrected: Remove leading/trailing whitespace before converting to int()
    current_pos_str_trimmed = current_pos_str.strip()

    if current_pos_str_trimmed == 'STOP':
        print("FMP is already stopped. Pause command is not effective.")
        PLAY_POSITION = 0  # Since it's stopped, reset position to 0
        return
    else:
        try:
            pause_position = int(current_pos_str_trimmed)  # Use the string after strip()
            print(f"Using current playing position for Pause: {pause_position} us")
        except ValueError:
            print(f"Could not parse CurrentPosition '{current_pos_str}' (trimmed to '{current_pos_str_trimmed}') to an integer. Using default pause position (10,000,000 us).")
            pause_position = 10000000  # Default value (10 seconds)
else:
    print("CurrentPosition not found in GetPlayingInfo response. Using default pause position (10,000,000 us).")
    pause_position = 10000000  # Default value (10 seconds)

command = f"Pause StopPosition={pause_position}"
responses = execute_command(send_mode, command, target_ip)
if responses and list(responses.values())[0].startswith("OK"):
    print(f"Pause command sent successfully. StopPosition: {pause_position} us")
    PLAY_POSITION = pause_position  # Save the paused position for next Play/Seek
else:
    print("Failed to send Pause command.")

def handle_stop(send_mode, target_ip):
    global PLAY_POSITION
    command = "Stop"  # Stop command has no parameters
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        first_ip = list(responses.keys())[0]
        response_str = responses[first_ip]
        if response_str.startswith("OK"):
            parsed_params = parse_response_params(response_str)
            current_pos_str = parsed_params.get('CurrentPosition')
            if current_pos_str:
                # First, trim leading/trailing whitespace
                current_pos_str_trimmed = current_pos_str.strip()

                if current_pos_str_trimmed == 'STOP':
                    print("Stop command sent successfully. FMP reported STOP status.")
                    PLAY_POSITION = 0  # Completely stopped, so position is 0
                else:
                    # Further remove all non-digit characters
                    # This reliably excludes invisible control characters or Unicode whitespace
                    current_pos_str_cleaned = re.sub(r'[^0-9]', '', current_pos_str_trimmed)

                    try:
                        PLAY_POSITION = int(current_pos_str_cleaned)  # Use the cleaned string
                        print(f"Stop command sent successfully. Stopped at position: {PLAY_POSITION} us")
                    except ValueError:
                        # Display a more detailed error message
                        print(f"Stop command sent successfully, but failed to parse CurrentPosition "
                              f"original:'{current_pos_str}' (trimmed:'{current_pos_str_trimmed}', cleaned:'{current_pos_str_cleaned}') "
                              f"to an integer. Setting PLAY_POSITION to 0.")
                        PLAY_POSITION = 0  # Reset on parse failure
            else:
                print("Stop command sent successfully, but CurrentPosition not found in response. Setting PLAY_POSITION to 0.")
                PLAY_POSITION = 0
        else:
            print(f"Failed to send Stop command: {response_str}")
            PLAY_POSITION = 0  # Reset on failure
    else:
        print("Failed to send Stop command or no response. Setting PLAY_POSITION to 0.")
        PLAY_POSITION = 0

def handle_seek(send_mode, target_ip):
    global PLAY_POSITION
```

```
seek_pos_input = input("Enter seek position in microseconds (e.g., 5000000 for 5 seconds): ")
try:
    seek_position = int(seek_pos_input)
    if seek_position < 0:
        print("Seek position cannot be negative.")
        return
except ValueError:
    print("Invalid seek position. Please enter a number.")
    return

command_params = [f"PlayPosition={seek_position}"]  # Place PlayPosition first

# Ask user whether to use SyncTime
use_sync_time = input("Use SyncTime? (y/n): ").lower()
if use_sync_time == 'y':
    if diff_time_us == 0:
        print("Warning: diff_time_us is 0. Running GetSystemTime first...")
        update_diff_time(send_mode, target_ip)
        # SyncTime = Current time (PC) + Marge time - Time difference
        sync_time_val = int(time.time() * 1000000) + marge_time_us - diff_time_us
        command_params.insert(0, f"SyncTime={sync_time_val}")  # Add SyncTime at the beginning

    # Construct the final command string
    # Seek Command: "Seek SyncTime=...&PlayPosition=..." or "Seek PlayPosition=..."
    final_command = "Seek " + "&".join(command_params)
    responses = execute_command(send_mode, final_command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print(f"Seek command sent successfully to position: {seek_position} us")
        PLAY_POSITION = seek_position  # Save the seeked position
    else:
        print(f"Failed to send Seek command to position: {seek_position} us")

def handle_skip(send_mode, target_ip):
    global PLAY_POSITION

    # Skip command does not have SyncTime parameter, so simply send the command
    final_command = "Skip"
    responses = execute_command(send_mode, final_command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print("Skip command sent successfully.")
        PLAY_POSITION = 0  # After skipping, usually returns to the beginning or the beginning of the next media
    else:
        print("Failed to send Skip command.")

def handle_disconnect_req(send_mode, target_ip):
    command = "DisconnectReq"
    responses = execute_command(send_mode, command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print("DisconnectReq sent successfully. FMP Timeline module should restart.")
    else:
        print("Failed to send DisconnectReq.")

def handle_reset_playlist(send_mode, target_ip):
    command = "ResetPlaylist"
    responses = execute_command(send_mode, command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print("ResetPlaylist sent successfully. Playlist reloaded.")
    else:
        print("Failed to send ResetPlaylist.")

def handle_set_sync_time(send_mode, target_ip):
    sync_time_offset_input = input("Enter SyncTime offset in seconds (0-30, default 3): ")
    try:
        sync_time_offset = int(sync_time_offset_input) if sync_time_offset_input else 3
        if not (0 <= sync_time_offset <= 30):
            print("SyncTime offset must be between 0 and 30 seconds.")
            return
    except ValueError:
        print("Invalid input. Please enter a number for SyncTime offset.")
        return

    command = f"SetSyncTime Time={sync_time_offset}"
    responses = execute_command(send_mode, command, target_ip)
    if responses and list(responses.values())[0].startswith("OK"):
        print(f"SetSyncTime command sent successfully. FMP internal offset set to {sync_time_offset} seconds.")
    else:
        print("Failed to send SetSyncTime command.")

def handle_get_playlist_info(send_mode, target_ip):
    debug_option = input("Include debug option (-d)? (y/n, default n): ").lower()
    command = "GetPlaylistInfo"
    if debug_option == 'y':
        command += " -d"

    responses = execute_command(send_mode, command, target_ip)
    if responses:
        print("\n--- GetPlaylistInfo Responses ---")
        for ip, response_str in responses.items():
```

```
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                for key, value in parsed_params.items():
                    if key.startswith("Playlist"):
                        print(f"  {key}: {value}")
            else:
                print(f"  Error: {response_str}")
    else:
        print("No FMP responded to GetPlaylistInfo or an error occurred.")

def handle_get_each_playlist_info(send_mode, target_ip):
    playlist_name_input = input(f"Enter playlist name to get info for (default: {PLAYLIST_NAME}): ") or PLAYLIST_NAME
    command = f"GetEachPlaylistInfo Playlist={playlist_name_input}"

    responses = execute_command(send_mode, command, target_ip)
    if responses:
        print(f"\n--- GetEachPlaylistInfo for '{playlist_name_input}' Responses ---")
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                print(f"  Playlist: {parsed_params.get('Playlist', 'N/A')}")
                print(f"  MediaNum: {parsed_params.get('Medianum', 'N/A')}")
                num_media = int(parsed_params.get('Medianum', 0))
                for i in range(1, num_media + 1):
                    print(f"    Media{i}: {parsed_params.get(f'Media{i}', 'N/A')} (Duration: {parsed_params.get(f'Duration{i}', 'N/A')}s)")
            else:
                print(f"  Error: {response_str}")
    else:
        print("No FMP responded to GetEachPlaylistInfo or an error occurred.")

def handle_get_playing_info(send_mode, target_ip):
    command = "GetPlayingInfo"
    responses = execute_command(send_mode, command, target_ip)
    if responses:
        print("\n--- GetPlayingInfo Responses ---")
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                parsed_params = parse_response_params(response_str)
                print(f"  Playlist: {parsed_params.get('Playlist', 'N/A')}")
                print(f"  CurrentPosition: {parsed_params.get('CurrentPosition', 'N/A')} us")
                print(f"  Loop: {parsed_params.get('Loop', 'N/A')}")
            else:
                print(f"  Error: {response_str}")
    else:
        print("No FMP responded to GetPlayingInfo or an error occurred.")

# --- Main Interaction Loop ---
if __name__ == "__main__":

    current_target_ip = DEFAULT_FMP_IP  # Current target IP
    current_send_mode = '1'  # Default is single FMP ('1': Single FMP, '2': Broadcast)

    while True:
        print("\n--- FMP Playback Control Tool ---")
        print(f"Current Target: {'Single FMP' if current_send_mode == '1' else 'Broadcast'} {'(' + current_target_ip + ')' if current_send_mode == '1' else ''}")
        print("-----------------------------------------")
        print("Playback Control Commands:")
        print("  1: GetSystemTime (Get FMP Time Difference)")
        print("  2: SelectPlayList (Select Playlist)")
        print("  3: Loop (Set Playlist Loop Status)")
        print("  4: Play (Start Playback)")
        print("  5: Pause (Pause Playback)")
        print("  6: Stop (Stop Playback)")
        print("  7: Seek (Seek to specific position)")
        print("  8: Skip (Skip to next media)")
        print("  9: DisconnectReq (Restart Timeline module)")
        print("  a: ResetPlaylist (Reload Playlist from storage)")
        print("  b: SetSyncTime (Set FMP internal SyncTime offset)")
        print("\nInformation Commands:")
        print("  c: GetPlaylistInfo (Get all Playlist names)")
        print("  d: GetEachPlaylistInfo (Get media info for a specific Playlist)")
        print("  e: GetPlayingInfo (Get current playback status)")
        print("\nUtility:")
        print("  f: Change Target (Single FMP / Broadcast)")
        print("  0: Exit")
        print("-----------------------------------------")

        choice = input("Enter command number/letter: ").lower()

        if choice == '1':
            handle_get_system_time(current_send_mode, current_target_ip)
        elif choice == '2':
            handle_select_playlist(current_send_mode, current_target_ip)
        elif choice == '3':
            handle_loop(current_send_mode, current_target_ip)
        elif choice == '4':
```

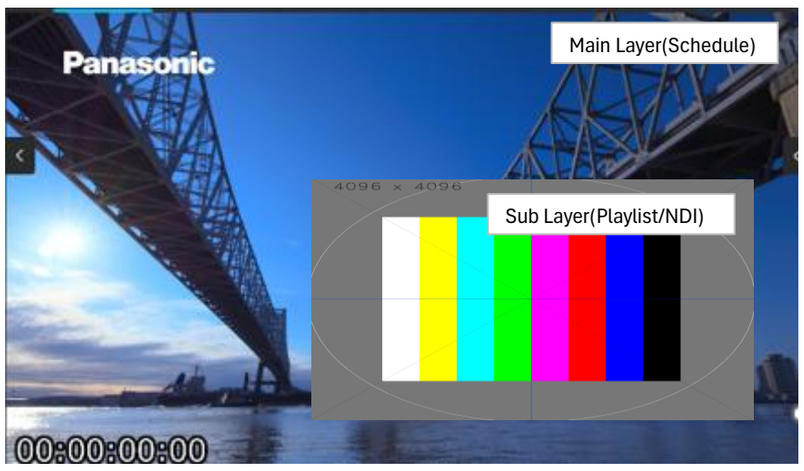
```
            handle_play(current_send_mode, current_target_ip)
        elif choice == '5':
            handle_pause(current_send_mode, current_target_ip)
        elif choice == '6':
            handle_stop(current_send_mode, current_target_ip)
        elif choice == '7':
            handle_seek(current_send_mode, current_target_ip)
        elif choice == '8':
            handle_skip(current_send_mode, current_target_ip)
        elif choice == '9':
            handle_disconnect_req(current_send_mode, current_target_ip)
        elif choice == 'a':
            handle_reset_playlist(current_send_mode, current_target_ip)
        elif choice == 'b':
            handle_set_sync_time(current_send_mode, current_target_ip)
        elif choice == 'c':
            handle_get_playlist_info(current_send_mode, current_target_ip)
        elif choice == 'd':
            handle_get_each_playlist_info(current_send_mode, current_target_ip)
        elif choice == 'e':
            handle_get_playing_info(current_send_mode, current_target_ip)
        elif choice == 'f':
            print("\n--- Change Target Mode ---")
            mode_choice = input("Select send mode (1: Single FMP, 2: Broadcast): ")
            if mode_choice == '1':
                current_send_mode = '1'
                current_target_ip = input(f"Enter target FMP IP address (default: {DEFAULT_FMP_IP}): ") or DEFAULT_FMP_IP
                print(f"Target set to Single FMP: {current_target_ip}")
            elif mode_choice == '2':
                current_send_mode = '2'
                current_target_ip = None  # Not applicable for broadcast in this context
                print(f"Target set to Broadcast (using {BROADCAST_IP})")
            else:
                print("Invalid mode selection. Keeping current mode.")

        elif choice == '0':
            print("Exiting program.")
            break
        else:
            print("Invalid command. Please enter again.")

    sock.close()
    print("Socket closed.")
```

■How to Use UDP Commands (PinP Playback Control)

● Overview

During scheduled playback by the Media Processor (FMP), interrupt playback using PinP (Picture in Picture)
can be performed via UDP command communication.
The media that can be used for interrupt playback is "Playlist/NDI".

※：For details on registering schedules and playlists, please refer to the VisualSoftwareSuite (VSS) manual.

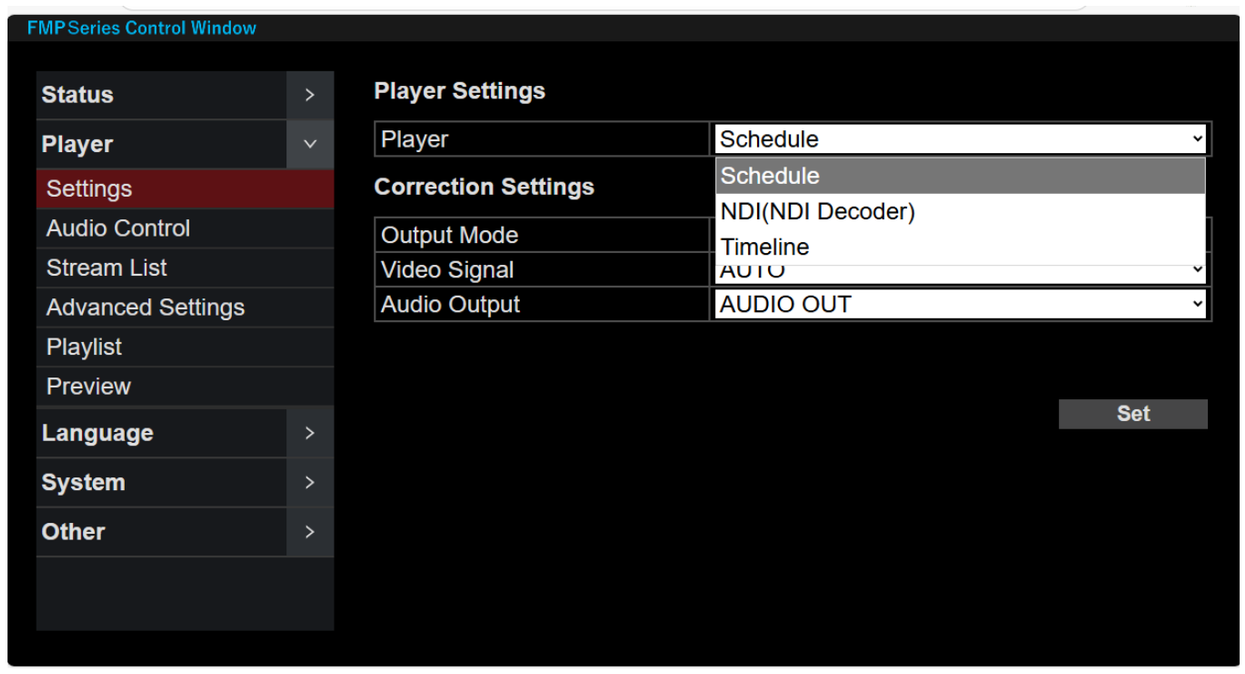


Main Layer (Schedule): Image of scheduled playback.

Sub Layer : Image of PinP (Playlist/NDI) interrupt playback.

● WEB (FMP Series Control Window) Settings

To use PinP interrupt playback, the Player setting must be "Schedule".



●PinP/Interrupt Restrictions and Conditions

Common:

Video Output: 30 fps on both the main and interrupt sides (60p content playback is possible, but reduce the frame rate)

Playlist:

HAP: Not supported on both the main and interrupt sides

Maximum of 150 Mbps on both the main and interrupt sides

NDI:

Resolution: Full HD (4K not supported)

Frame Rate: 30/25

Bitrate: Maximum of 100 Mbps recommended

If the resolution, frame rate, or bitrate exceeds the above, playback is possible, but stuttering, stoppages, or synchronization errors may occur.

● UDP Command Specifications

FMP Connection Conditions

Player: Schedule is running and waiting on the following socket.

Address: Address set in FMP

Listening port number: 65432

Source address: INADDR_ANY

Maximum send/receive size: 65536 bytes

Operation Command List

Command	説明
pinpPlay	**Start interrupt playback (PinP)**
pinpStop	**Stop interrupt playback (PinP)**
GetSystemTime	**Time adjustment between UDP command sender and FMP**

Information Command List

Command	説明
GetStreamList	**Get list of NDI streams**

	Start interrupt playback (PinP)

Command	**pinpPlay**

Description

Command to start PinP playback.
Only works when player setting is schedule.

Request

Parameter	Always	Description	Remarks
PlayerType	○	Player type for interrupt(Timeline, NDI)	
Source	○	Player type for interrupt：When PlayerType=Timeline (playlist name), when NDI (NDI source name)	
Scale		Output video size[%](0 - 100)	Default is[100]
PositionX		Output video display position [Xcoordinate](0 - 100)	Default is[0]
PositionY		Output video display position [Ycoordinate](0 - 100)	Default is[0]
Audio		Output audio selection(Main, Sub)	Select output audio. Main: Schedule audio, Sub: PinP audio. Default is [Sub].
Loop		Loop setting(ON, OFF)	Only when PlayerType is Timeline. Default is [ON]. If [OFF], PinP playback ends after one playlist playback.
Wait		Wait time until playback starts [Sec](0 - 30)	Set the time required for internal decoding processing. Set at least 3 seconds. Default is [0], playback starts when internal preparation is complete. Set sufficient wait time for synchronization with multiple units. SyncTime takes precedence if set.
SyncTime		Playback start synchronization time (Linux time + μ seconds)	Linux time format (including μ seconds after decimal point) multiplied by 1,000,000. To ensure FMP internal processing, specify a future time of at least 3 seconds. Also, include the time difference from GetSystemTime. If accurate synchronization cannot be achieved with "Wait" setting, use this parameter to set the synchronized playback start time.

Request Example

・When performing PinP playback (Timeline) in the upper right 1/4 of the screen (audio from Main screen, PinP loop playback)
pinpPlay PlayerType=Timeline&Source=Playlist01&Scale=50&PositionX=50&PositionY=50&Audio=Main&Loop=ON
・When performing PinP (interrupt) playback (NDI) on the entire screen (when starting playback 5 seconds later with multi-unit synchronization)
pinpPlay PlayerType=NDI&Source=Device1&Wait=5

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

OK

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="

On Failure

NG Invalid parameter

※：NG may be due to the following reasons.
・Invalid parameter
・Playback with a player other than Schedule (NDI, Timeline)

	Stop interrupt playback (PinP)

Command	**pinpStop**

Description

Command to stop PinP playback.
Only works when player setting is schedule.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
pinpStop
```

Return Value

Parameter	Always	Description	Remarks
OK/NG	○	Result	

Return Example

On Success

```
OK
```

On Failure

```
NG

※：NG may be due to the following reasons.
  ・Not in PinP playback state.
```

	Time adjustment between UDP command execution side and FMP side

Command	**GetSystemTime**

Description

To determine the time difference between the internal clock on the PC executing the UDP command and the FMP side.

The time difference should be calculated in microseconds, and this value should be added to the parameter "playback start synchronization time" of the subsequent UDP command.

Request

Parameter	Always	Description	Remarks
CurrentTimeVss	○	UDP Commands execution time (LinuxTime + usec)	Linux time format (including usec after the decimal point) multiplied by 1000000

Request Example

GetSystemTime CurrentTimeVss=1715237197598000

*1: The separator between UDP commands and parameters is " " (space), and the value of the parameter is after "=".
*2: 1715237197598000＝2024/05/09 06:46:37.598000

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
CurrentTimeVss	○	Execution time set at the time of request	
CurrentTimeFmp	○	FMP Time(Linux time + usec)	Linux time format (including usec after the decimal point) multiplied by 1000000

Return Example

On Success

OK CurrentTimeVss=1715237197598000&CurrentTimeFmp=1715237196004200

※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
※２：1715237197598000＝2024/05/09 06:46:37.598000
　　1715237196004200＝2024/05/09 06:46:36.004200

On Failure

None

	Get NDI stream list

Command	**GetStreamList**

Description

Command to get NDI stream list.

Request

Parameter	Always	Description	Remarks
None			

Request Example

```
GetStreamList
```

Return Value

Parameter	Always	Description	Remarks
OK	○	Result	Always [OK] is returnd
StreamNum	○	Number of streams	
Source1		1st source	※ encodeURIComponent
Status1		1st playback status(0:Stopped 1:Playing)	
Source2		2nd source	※ encodeURIComponent
Status2		2nd playback status(0:Stopped 1:Playing)	
...			
SourceN		Nth source	※ encodeURIComponent
StatusN		Nth playback status(0:Stopped 1:Playing)	

Return Example

On Success

```
OK StreamNum=2&Source1=xxx&Status1=0&Source2=yyy&Status2=1


※：The separator between the execution result and parameters is " " (space), the separator between parameters is "&", and the value of a parameter follows "="
```

On Failure

```
NG Not Support

※：NG may be due to the following reasons.
   ・NDI not selected, VSS Timeline is active (Not Support)
```

●Sample code for PinP Control（python）

This is a sample code for PinP control.

Please change 'DEFAULT_FMP_IP' and 'BROADCAST_IP' according to the IP address of the FMP you are using (local IP).

```
import socket
import time
import re

# --- Constant Settings ---
M_SIZE = 1024  # Receive buffer size

# FMP address and target port number
FMP_PORT = 65432

# Default IP for single FMP mode
DEFAULT_FMP_IP = '192.168.0.11' # Change this to your actual FMP IP

# Broadcast address (usually .255 for IPv4 /24 subnet)
# Adjust if your subnet mask is different
BROADCAST_IP = '192.168.0.255' # Example broadcast IP for a /24 subnet

# --- UDP Socket Initialization ---
# This socket will be reused, so ensure it's not bound to a specific IP for broadcast if needed
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1) # Enable broadcast mode for this socket
sock.settimeout(10) # Default timeout for single FMP operations

# --- Global Variables for Playback State ---
diff_time_us = 0 # Time difference with FMP (in microseconds)
marge_time_us = 3000000 # FMP processing buffer time (3 seconds = 3,000,000 microseconds) - 3 seconds or more recommended

# --- Helper Functions for sending commands ---

def send_command_single_fmp(target_ip, command_str):
    """
    Sends the specified UDP command string to a single FMP and receives a response.
    Returns a dictionary with FMP IP as key and response string as value.
    """
    target_address = (target_ip, FMP_PORT)
    print(f"Sending to {target_address}: {command_str}")
    try:
        sock.settimeout(10) # Ensure timeout is set for single FMP
        sock.sendto(command_str.encode('utf-8'), target_address)
        rx_message, addr = sock.recvfrom(M_SIZE)
        decoded_message = rx_message.decode(encoding='utf-8')
        print(f"Received from {addr}: {decoded_message}")
        return {addr[0]: decoded_message} # Return as a dictionary for consistency
    except socket.timeout:
        print(f"Error: Socket timeout when sending to {target_address}. FMP might not be responding.")
        return {target_ip: "ERROR: TIMEOUT"}
    except Exception as e:
        print(f"Error sending/receiving data to {target_address}: {e}")
        return {target_ip: f"ERROR: EXCEPTION ({e})"}

def send_command_broadcast(command_str, timeout_seconds=3, expected_fmp_count=None):
    """
    Sends the specified UDP command string via broadcast and collects responses from multiple FMPs.
    Returns a dictionary with FMP IP as key and response string as value for each FMP.
    """
    broadcast_address = (BROADCAST_IP, FMP_PORT)
    print(f"Sending broadcast to {broadcast_address}: {command_str}")

    responses = {}
    sock.settimeout(0.1) # Shorter timeout for collecting multiple responses
    start_time = time.time()

    try:
        sock.sendto(command_str.encode('utf-8'), broadcast_address)

        while time.time() - start_time < timeout_seconds:
            try:
                rx_message, addr = sock.recvfrom(M_SIZE)
                decoded_message = rx_message.decode(encoding='utf-8')
                print(f"Received from {addr}: {decoded_message}")
                responses[addr[0]] = decoded_message # Store response with IP as key

                if expected_fmp_count is not None and len(responses) >= expected_fmp_count:
```

```
                print(f"Collected {len(responses)} responses, which is the expected count.")
                break

        except socket.timeout:
            # No more packets in the buffer for this short timeout
            pass
        except Exception as e:
            print(f"Error receiving data in broadcast loop: {e}")
            break

    if not responses:
        print("No FMP responses received within the timeout period.")

    return responses

except Exception as e:
    print(f"Error sending broadcast or setting up socket: {e}")
    return {}
finally:
    # Reset timeout to default for single FMP operations
    sock.settimeout(10)

def execute_command(send_mode, command_str, target_ip=None):
    """Wrapper function to execute command based on send_mode."""
    if send_mode == '1': # Single FMP
        if target_ip is None: # Should ideally be passed when send_mode is '1'
            target_ip = input(f"Enter target FMP IP address (default: {DEFAULT_FMP_IP}): ") or DEFAULT_FMP_IP
        return send_command_single_fmp(target_ip, command_str)
    elif send_mode == '2': # Broadcast
        broadcast_timeout = float(input("Enter broadcast response collection timeout in seconds (e.g., 3.0): ") or 3.0)
        expected_count_input = input("Enter expected number of FMPs (leave blank if unknown): ")
        expected_count = int(expected_count_input) if expected_count_input else None
        return send_command_broadcast(command_str, timeout_seconds=broadcast_timeout, expected_fmp_count=expected_count)
    else:
        print("Invalid send mode. Please choose 1 or 2.")
        return {}

def parse_response_params(response_str):
    """
    Parses FMP's OK response string into a dictionary of parameters.
    Example: "OK Player=1&ByVss=0&Content=abc.mp4" -> {"Player": "1", "ByVss": "0", "Content": "abc.mp4"}
    """
    params = {}
    if not response_str.startswith("OK"):
        return params

    # Remove "OK " prefix and split by "&"
    param_pairs = response_str[3:].split('&')
    for pair in param_pairs:
        if '=' in pair:
            key, value = pair.split('=', 1)
            params[key] = value
        else:
            params[pair] = True # For parameters like "-d" (debug option) without value
    return params

# --- Command Specific Helper Functions ---

def update_diff_time(send_mode, target_ip):
    """
    Executes GetSystemTime command and updates the time difference (diff_time_us) with the FMP.
    For broadcast mode, it uses the time from the first responding FMP.
    """
    global diff_time_us
    start_u = int(time.time() * 1000000)
    command = f"GetSystemTime CurrentTimeVss={start_u}"

    responses = execute_command(send_mode, command, target_ip)

    if responses:
        first_ip = list(responses.keys())[0]
        response = responses[first_ip]

        if response.startswith("OK"):
            fmp_time_match = re.search(r'CurrentTimeFmp=(\d+)', response)
            if fmp_time_match:
                fmp_time = int(fmp_time_match.group(1))
                diff_time_us = start_u - fmp_time
                print(f"System time difference (PC_Vss - FMP_Fmp) from {first_ip}: {diff_time_us} us")
```

```
                return diff_time_us
            else:
                print(f"Error: CurrentTimeFmp not found in GetSystemTime response from {first_ip}.")
                return 0
        else:
            print(f"GetSystemTime command failed for {first_ip}: {response}")
            return 0
    print("No FMP responded to GetSystemTime or an error occurred.")
    return 0

def get_ndi_stream_list(send_mode, target_ip):
    """
    Executes GetStreamList command and collects a list of available NDI streams.
    In broadcast mode, it integrates lists from all FMPs.
    """
    command = "GetStreamList"
    responses = execute_command(send_mode, command, target_ip)

    all_ndi_sources = set() # Use a set to avoid duplicates

    if responses:
        print("\n--- NDI Stream List Responses ---")
        for ip, response_str in responses.items():
            print(f"FMP ({ip}): {response_str}")
            if response_str.startswith("OK"):
                stream_num_match = re.search(r'StreamNum=(\d+)', response_str)
                if stream_num_match:
                    stream_num = int(stream_num_match.group(1))
                    for i in range(1, stream_num + 1):
                        source_match = re.search(fr'Source{i}=([^&]+)', response_str)
                        if source_match:
                            all_ndi_sources.add(source_match.group(1))
            else:
                print(f"  Error: {response_str}")
    else:
        print("No FMP responded to GetStreamList or an error occurred.")

    return sorted(list(all_ndi_sources)) # Return as a sorted list

# --- Main Command Handling Functions ---

def handle_get_system_time(send_mode, target_ip):
    global diff_time_us
    diff_time_us = update_diff_time(send_mode, target_ip)
    print(f"Current System Time Difference (diff_time_us): {diff_time_us} us")

def handle_pinp_play(send_mode, target_ip):
    print("\n--- PinP Playback Start Settings ---")

    # Select player type
    player_type_choice = input("Select PinP Player Type (1: Timeline, 2: NDI): ")
    player_type = ""
    source_name = ""

    if player_type_choice == '1':
        player_type = "Timeline"
        # Set default playlist name to 'Playlist01'
        source_name = input("Enter playlist name to play (default: Playlist01): ") or "Playlist01"
    elif player_type_choice == '2':
        player_type = "NDI"
        print("\nFetching NDI source list...")
        ndi_sources = get_ndi_stream_list(send_mode, target_ip) # Pass send_mode and target_ip

        if not ndi_sources:
            print("No available NDI sources found.")
            return

        print("\n--- Available NDI Sources ---")
        for i, src in enumerate(ndi_sources):
            print(f"{i+1}: {src}")

        source_index_str = input("Select the number of the NDI source to play: ")
        try:
            source_index = int(source_index_str) - 1
            if 0 <= source_index < len(ndi_sources):
                source_name = ndi_sources[source_index]
            else:
                print("Invalid number.")
                return
```

```
    except ValueError:
        print("Invalid input. Please enter a number.")
        return
else:
    print("Invalid player type selection.")
    return

# Build PinP command base
pinp_command_parts = [f"pinpPlay PlayerType={player_type}", f"Source={source_name}"]

# Check if SyncTime should be used
use_sync_time_flag = False
use_sync_time_input = input("Use SyncTime? (y/n): ").lower()
if use_sync_time_input == 'y':
    use_sync_time_flag = True
    if diff_time_us == 0:
        print("Warning: diff_time_us is 0. Running GetSystemTime first...")
        update_diff_time(send_mode, target_ip) # Pass send_mode and target_ip
else:
    # If not using SyncTime, ask for Wait parameter
    wait_time = input("Wait time (seconds, 0-30, leave blank to omit): ")
    if wait_time:
        try:
            wait_time_int = int(wait_time)
            if 0 <= wait_time_int <= 30:
                pinp_command_parts.append(f"Wait={wait_time_int}")
            else:
                print("Wait time must be between 0 and 30 seconds.")
                return
        except ValueError:
            print("Invalid Wait time.")
            return

# Other optional parameters
scale = input("Scale (0-100, leave blank to omit, default 100): ")
pos_x = input("PositionX (0-100, leave blank to omit, default 0): ")
pos_y = input("PositionY (0-100, leave blank to omit, default 0): ")
audio = input("Audio (Main/Sub, leave blank to omit, default Sub): ").capitalize()
loop = input("Loop (ON/OFF, leave blank to omit, default ON for Timeline, OFF for NDI): ").upper()

if scale:
    pinp_command_parts.append(f"Scale={scale}")
if pos_x:
    pinp_command_parts.append(f"PositionX={pos_x}")
if pos_y:
    pinp_command_parts.append(f"PositionY={pos_y}")
if audio in ["Main", "Sub"]:
    pinp_command_parts.append(f"Audio={audio}")
if loop in ["ON", "OFF"]:
    pinp_command_parts.append(f"Loop={loop}")

# Calculate and add SyncTime just before sending the command
if use_sync_time_flag:
    # SyncTime = Current Time (PC) + Marge Time - Time Difference
    sync_time_val = int(time.time() * 1000000) + marge_time_us - diff_time_us
    pinp_command_parts.append(f"SyncTime={sync_time_val}")

pinp_command_final = "&".join(pinp_command_parts)
responses = execute_command(send_mode, pinp_command_final, target_ip)
if responses and all(r.startswith("OK") for r in responses.values()):
    print("pinpPlay command sent successfully.")
else:
    print("Failed to send pinpPlay command.")


def handle_pinp_stop(send_mode, target_ip):
    command = "pinpStop"
    responses = execute_command(send_mode, command, target_ip)
    if responses and all(r.startswith("OK") for r in responses.values()):
        print("pinpStop command sent successfully.")
    else:
        print("Failed to send pinpStop command.")

# --- Main Interaction Loop ---
if __name__ == "__main__":

    current_target_ip = DEFAULT_FMP_IP # Current target IP
    current_send_mode = '1' # Default is single FMP ('1': Single FMP, '2': Broadcast)
```

```
while True:
    print("\n--- FMP PinP Playback Control Tool ---")
    print(f"Current Target: {'Single FMP' if current_send_mode == '1' else 'Broadcast'} {'(' + current_target_ip + ')' if current_send_mode == '1' else ''}")
    print("----------------------------------------")
    print("PinP Control Commands:")
    print("  1: pinpPlay (Start PinP Playback)")
    print("  2: pinpStop (Stop PinP Playback)")
    print("  3: GetSystemTime (Synchronize Time)")
    print("\nUtility:")
    print("  f: Change Target (Single FMP / Broadcast)")
    print("  0: Exit")
    print("----------------------------------------")

    choice = input("Enter command number/letter: ").lower()

    if choice == '1':
        handle_pinp_play(current_send_mode, current_target_ip)
elif choice == '2':
    handle_pinp_stop(current_send_mode, current_target_ip)
elif choice == '3':
    handle_get_system_time(current_send_mode, current_target_ip)
elif choice == 'f':
    print("\n--- Change Target Mode ---")
    mode_choice = input("Select send mode (1: Single FMP, 2: Broadcast): ")
    if mode_choice == '1':
        current_send_mode = '1'
        current_target_ip = input(f"Enter target FMP IP address (default: {DEFAULT_FMP_IP}): ") or DEFAULT_FMP_IP
        print(f"Target set to Single FMP: {current_target_ip}")
    elif mode_choice == '2':
        current_send_mode = '2'
        current_target_ip = None # Not applicable for broadcast in this context
        print(f"Target set to Broadcast (using {BROADCAST_IP})")
    else:
        print("Invalid mode selection. Keeping current mode.")

elif choice == '0':
    print("Exiting program.")
    break
else:
    print("Invalid command. Please enter again.")

sock.close()
print("Socket closed.")
```