UDP コマンド ユーザーマニュアル

対象機種 ET-FMP50
ET-FMP20
ET-SBFMP10

Panasonic

(2026-03)

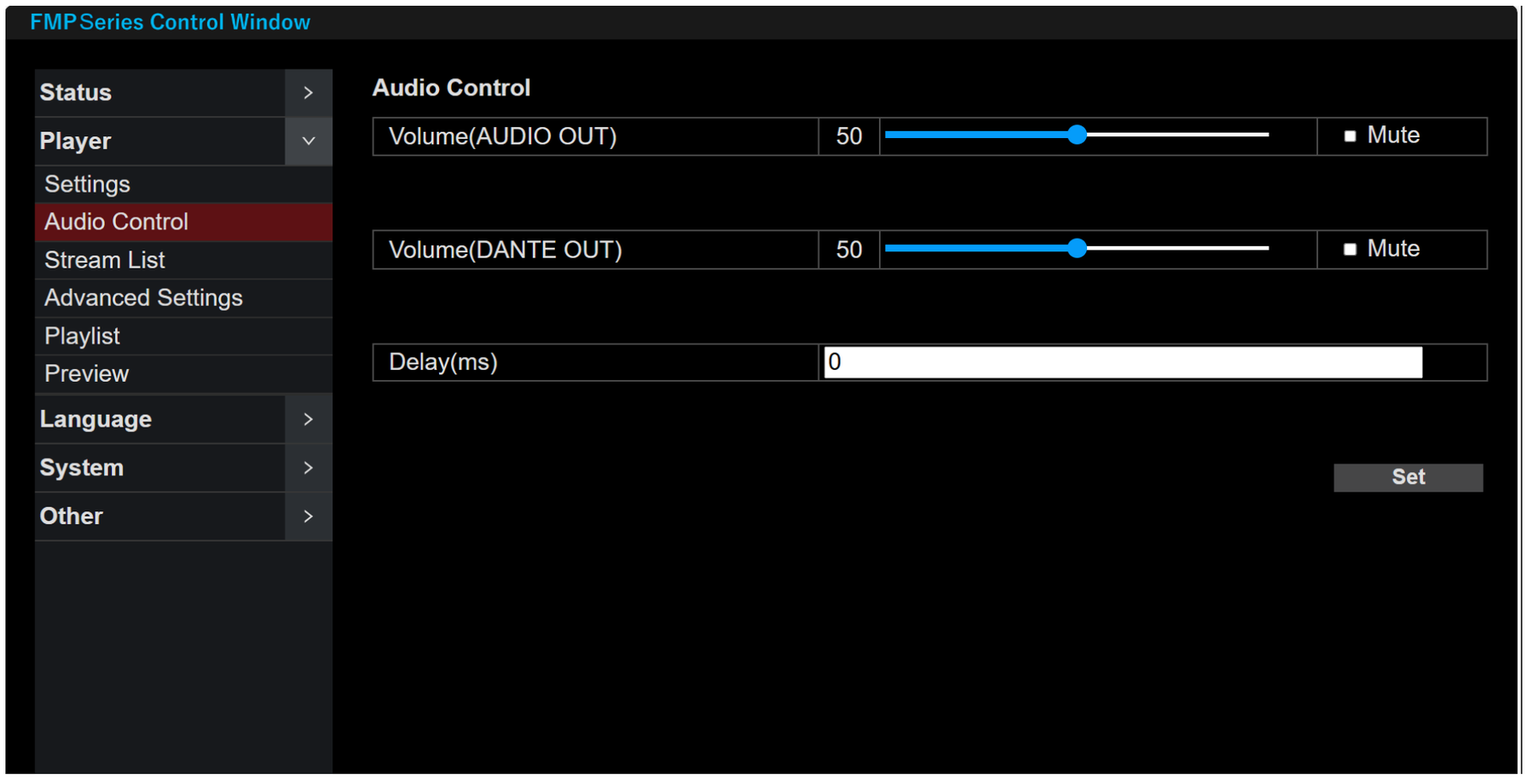
■UDPコマンド使用方法(設定制御)

●概要

メディアプロセッサ（FMP）のWebページで行う各種設定をUDPコマンド通信でおこうことができます。

※：一部の再生操作系やファイル操作をともなう設定には非対応です。

例）Webページで音量変更やMute操作を実施する場合、UDPコマンドを使ってWebページを開くことなく音量変更やMute操作が可能です。



●UDPコマンド仕様

FMP側接続条件

FMP起動し、以下のソケットで待ち受けしている。

アドレス：**FMPに設定したアドレス**

待ち受けポート番号：**65430**

接続元アドレス：**INADDR_ANY**

最大送受信サイズ：**65536byte**

操作系コマンド一覧

コマンド	説明
SetPlayerProperty	**プレイヤー情報の設定**
SetAudioProperty	**オーディオ情報の設定**
SetAdvancedProperty	**高度な設定情報の設定**
PlayStream	**NDIストリーミング再生の開始**
StopStream	**ストリーミング再生の停止**
SetAutoPlayProperty	**自動再生ストリーム情報の設定**
SetPlaylistSyncProperty	**プレイリスト同期情報の設定**
SetLanguage	**表示言語の設定**
SetHostName	**ホスト名の設定**
SetLEDProperty	**POWER LED情報の設定**
SetNetworkProperty	**ネットワーク情報の設定**
SetTimeZone	**タイムゾーンの設定**
SetDatetimeProperty	**日時情報の設定**
SetAccountProperty	**Webアカウント情報の変更**
execInitialize	**システムの初期化**
Reboot	**システム再起動**

情報系コマンド一覧

コマンド	説明
GetPlayerState	**プレイヤー再生状況の取得**
GetPlayerProperty	**プレイヤー情報の取得**
GetAudioProperty	**オーディオ情報の取得**
GetAdvancedProperty	**高度な設定情報の取得**
GetStreamList	**NDIストリーム一覧の取得**
GetAutoPlayProperty	**自動再生ストリーム情報の取得**
GetPlaylistSyncProperty	**プレイリスト同期情報の取得**
GetLanguage	**表示言語の取得**
GetHostName	**ホスト名の取得**
GetLEDProperty	**POWER LED情報の取得**
GetNetworkProperty	**ネットワーク情報の取得**
GetDatetimeProperty	**日時情報の取得**

プレイヤー転送コマンド

以下の設定制御コマンドを使って、後述の「再生制御コマンド」「PinP制御コマンド」を使用することが可能です。
ProxyPlayerコマンドの後にスペースを入れて、その後に使用したいコマンド（再生制御、PinP制御）を記載することで動作します。

コマンド	説明
ProxyPlayer	**プレイヤーへのプロキシ** Player（Schedule / NDI / Timeline）へUDPコマンドを転送する、 Player側のUDPポートが「65432」です。 Playerからの返事をそのまま返事する

リクエスト例：

ProxyPlayer GetSystemTime CurrentTimeVss=1715237197598000

リスポンス：

- UDP転送失敗(UDP transfer failed)　　※Player未起動の場合、UPD送信タイムアウト（10s）
- Playerからの戻り値

OK CurrentTimeVss=1715237197598000&CurrentTimeFmp=1715237196004200

共通エラー

1) 不正なコマンド(NG Unknown command)
2) Webアカウントが存在しない(NG Account not set up)

	プレイヤー再生状況の取得

コマンド	GetPlayerState

内容

プレイヤー再生状態を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetPlayerState

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
Player	○	選択中のプレイヤー (1:Schedule 2:NDI(NDI Decoder) 3:Timeline)	
ByVss	○	VSSにてTimeline起動 (0:非VSS起動 1:VSS起動)	
Content		コンテンツ名 / 信号情報(NDI)	停止中の場合設定しない ※ encodeURIComponent

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「 」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	プレイヤー情報の取得

コマンド	GetPlayerProperty

内容

プレイヤー情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetPlayerProperty

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	本コマンドは「OK」のみ
Player	○	選択中のプレイヤー (1:Schedule 2:NDI(NDI Decoder) 3:Timeline)	
OutputMode	○	出力モード(1, 4)	
VideoSignal	○	映像信号 (0:AUTO 1:3840x2160/60p 2:3840x2160/50p 3:1080/60p 4:1080/50p)	
AudioOutput	○	音声出力 (0:HDMI OUT 1:AUDIO OUT 2:DANTE)	
Screen1IPAddress		画面 1 IPアドレス	出力モードが"4"の場合のみ
Screen2IPAddress		画面 2 IPアドレス	出力モードが"4"の場合のみ
Screen3IPAddress		画面 3 IPアドレス	出力モードが"4"の場合のみ
Screen4IPAddress		画面 4 IPアドレス	出力モードが"4"の場合のみ

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	プレイヤー情報の設定

コマンド	**SetPlayerProperty**

内容

プレイヤー情報を設定するためのコマンド

※処理に時間がかかる為応答まで最大30秒かかる場合があります

リクエスト

パラメータ名	必須	内容	備考
Player	○	プレイヤー (1:Schedule 2:NDI(NDI Decoder) 3:Timeline)	
OutputMode	○	出力モード (1, 4)	SDM(SBFMP10)の場合：対象外
VideoSignal	○	映像信号 (0:AUTO 1:3840x2160/60p 2:3840x2160/50p 3:1080/60p 4:1080/50p)	
AudioOutput	○	音声出力 (0:HDMI OUT 1:AUDIO OUT 2:DANTE)	
Screen1IPAddress		画面 1 IPアドレス	出力モードが"4"の場合、設定必要
Screen2IPAddress		画面 2 IPアドレス	出力モードが"4"の設定、設定必要
Screen3IPAddress		画面 3 IPアドレス	出力モードが"4"の設定、設定必要
Screen4IPAddress		画面 4 IPアドレス	出力モードが"4"の設定、設定必要

リクエスト例

```
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
```

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

```
OK
```

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

```
NG Invalid parameter
```

※：NGは、以下の要因が考えられます。
- パラメータ不正(Invalid parameter)

	オーディオ情報の取得

コマンド	GetAudioProperty

内容

オーディオ情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetAudioProperty

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
HdmiVolume	○	音量(0 - 100)	SDM(SBFMP10)の場合：対象外
HdmiMute	○	消音(true, false)	SDM(SBFMP10)の場合：対象外
AnalogVolume	○	音量(0 - 100)	
AnalogMute	○	消音(true, false)	
DanteVolume	○	音量(0 - 100)	
DanteMute	○	消音(true, false)	
Delay	○	音声遅延時間(0 - 171)	

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	オーディオ情報の設定

コマンド	**SetAudioProperty**

内容

オーディオ情報を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
HdmiVolume	○	音量(0 - 100)	SDM(SBFMP10)の場合：対象外
HdmiMute	○	消音(true, false)	SDM(SBFMP10)の場合：対象外
AnalogVolume	○	音量(0 - 100)	
AnalogMute	○	消音(true, false)	
DanteVolume	○	音量(0 - 100)	
DanteMute	○	消音(true, false)	
Delay	○	音声遅延時間(0 - 171)	

リクエスト例

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

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
　・パラメータ不正(Invalid parameter)

	高度な設定情報の取得

コマンド	GetAdvancedProperty

内容

高度な設定情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetAdvancedProperty

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
AntiAliasing	○	アンチエイリアス (0.00:OFF 0.25:Low 0.50:High)	

リターン例

成功時

OK AntiAliasing=0.50

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	高度な設定情報の設定

コマンド	**SetAdvancedProperty**

内容

高度な設定情報を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
AntiAliasing	○	アンチエイリアス (0.00:OFF 0.25:Low 0.50:High)	

リクエスト例

```
SetAdvancedProperty AntiAliasing=0.50
```

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

```
OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります
```

失敗時

```
NG Invalid parameter

※：NGは、以下の要因が考えられます
   ・パラメータ不正(Invalid parameter)
```

	NDIストリーム一覧の取得

コマンド	GetStreamList

内容

NDIストリーム一覧を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

```
GetStreamList
```

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
StreamNum	○	ストリーム数	
Source1		１つ目のソース	※ encodeURIComponent
Status1		１つ目の再生状態(0:Stopped 1:Playing)	
Source2		２つ目のソース	※ encodeURIComponent
Status2		２つ目の再生状態(0:Stopped 1:Playing)	
…			
SourceN		Ｎつ目のソース	※ encodeURIComponent
StatusN		Ｎつ目の再生状態(0:Stopped 1:Playing)	

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

```
NG Not Support
```

※：NGは、以下の要因が考えられます
・NDIが選択されていない、VSSでTimeline起動中(Not Support)

	NDIストリーミング再生の開始

コマンド	PlayStream

内容

NDIストリーミング再生を開始するためのコマンド

※処理に時間がかかる為応答まで最大10秒かかる場合があります

リクエスト

パラメータ名	必須	内容	備考
Source	○	ソース	※ decodeURIComponent

リクエスト例

PlayStream Source=xxx

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
- パラメータ不正(Invalid parameter)
- NDIが選択されていない、VSSでTimeline起動中(Not Support)
- ソースが存在しない(Execution failed)

	ストリーミング再生の停止

コマンド	StopStream

内容

ストリーミング再生を停止するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

StopStream

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Not Support

※：NGは、以下の要因が考えられます
　・NDIが選択されていない、VSSでTimeline起動中(Not Support)

	自動再生ストリーム情報の取得

コマンド	GetAutoPlayProperty

内容

自動再生ストリーム情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetAutoPlayProperty

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
AutoPlayStatus	○	自動再生有効状態(true, false)	
AutoPlaySource		自動再生ストリームソース	※ encodeURIComponent

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Not Support

※：NGは、以下の要因が考えられます
　・NDIが選択されていない、VSSでTimeline起動中(Not Support)

	自動再生ストリーム情報の設定

コマンド	**SetAutoPlayProperty**

内容

自動再生ストリーム情報を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
AutoPlayStatus	○	自動再生有効状態(true, false)	
AutoPlaySource	○	ソース	※ decodeURIComponent

リクエスト例

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

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
- パラメータ不正(Invalid parameter)
- NDIが選択されていない、VSSでTimeline起動中(Not Support)
- ソースが存在しない(Execution failed)

	プレイリスト同期情報の取得

コマンド	GetPlaylistSyncProperty

内容

プレイリスト同期情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetPlaylistSyncProperty

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
PlaylistSync	○	プレイリストの同期情報(true, false)	
PlayContolSync	○	再生制御の同期情報(true, false)	

リターン例

成功時

OK PlaylistSync=false&PlayContolSync=false

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	プレイリスト同期情報の設定

コマンド	**SetPlaylistSyncProperty**

内容

プレイリスト同期情報を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
PlaylistSync	○	プレイリストの同期情報(true, false)	
PlayContolSync	○	再生制御の同期情報(true, false)	

リクエスト例

SetPlaylistSyncProperty PlaylistSync=false&PlayContolSync=true

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
　・パラメータ不正(Invalid parameter)

	表示言語の取得

コマンド	GetLanguage

内容

表示言語を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetLanguage□

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
Language	○	表示言語(0:English 1:Japanese)	

リターン例

成功時

OK Language=0

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	表示言語の設定

コマンド	SetLanguage

内容

表示言語を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
Language	○	表示言語(0:English 1:Japanese)	

リクエスト例

SetLanguage Language=0

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
　・パラメータ不正(Invalid parameter)

	ホスト名の取得

コマンド	GetHostName

内容

ホスト名を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetHostName

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
Hostname	○	ホスト名	

リターン例

成功時

OK Hostname=NAME3821

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	ホスト名の設定

コマンド	**SetHostName**

内容

ホスト名を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
Hostname	○	ホスト名	

リクエスト例

```
SetHostName Hostname=NAME3821
```

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

```
OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります
```

失敗時

```
NG Invalid parameter

※：NGは、以下の要因が考えられます
   ・パラメータ不正(Invalid parameter)
```

	POWER LED情報の取得

コマンド	**GetLEDProperty**

内容

POWER LED情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetLEDProperty

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
Mode	○	モード(0:Nomal 1:Off)	
Notification	○	通知(0:Disable 1:Enable)	

リターン例

成功時

OK Mode=0&Notification=1

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	POWER LED情報の設定

コマンド	**SetLEDProperty**

内容

POWER LED情報を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
Mode	○	モード(0:Normal 1:Off)	
Notification	○	通知(0:Disable 1:Enable)	

リクエスト例

SetLEDProperty Mode=0&Notification=1

※：モードが「0:Normal」の場合、Notificationの設定値に関わらず「1:Enable」となります

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
　・パラメータ不正(Invalid parameter)

	ネットワーク情報の取得

コマンド	GetNetworkProperty

内容

ネットワーク情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetNetworkProperty

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
DHCP	○	DHCP(true:ON, false:OFF)	
IPAddress		IPアドレス	DHCPが"false"の場合
SubnetMask		サブネットマスク	DHCPが"false"の場合
Gateway		デフォルトゲートウェイ	DHCPが"false"の場合
DNS		DNS	

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	ネットワーク情報の設定

コマンド	**SetNetworkProperty**

内容

ネットワーク情報を設定するためのコマンド

※処理に時間がかかる為応答まで最大20秒かかる場合があります

リクエスト

パラメータ名	必須	内容	備考
DHCP	○	DHCP(true:ON, false:OFF)	
IPAddress		IPアドレス	DHCPが"false"の場合設定する
SubnetMask		サブネットマスク	DHCPが"false"の場合設定する
Gateway		デフォルトゲートウェイ	DHCPが"false"の場合設定する
DNS		DNS	

リクエスト例

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

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：IP変更の場合、ネットワーク再起動でリスポンス受信できません
※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
・パラメータ不正(Invalid parameter)
・設定失敗(Execution failed)

	日時情報の取得

コマンド	GetDatetimeProperty

内容

日時情報を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetDatetimeProperty

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
TimezoneLocation	○	タイムゾーン 地域	
TimezoneCity	○	タイムゾーン 都市	
CurrentTime	○	現在時刻(YYYYMMDDhhmmss)	
SyncProtocol	○	同期プロトコル(0:NTP 1:SoftwareSync)	
DomainNo		ドメイン番号(0 - 127)	同期プロトコルがSoftwareSyncの場合
OrderPs		プライマリー / セカンダリー (0:Primary 1:Secondary)	同期プロトコルがSoftwareSyncの場合
NTPSynchronization		NTP同期(true:ON false:OFF)	同期プロトコルがNTPの場合 同期プロトコルがSoftwareSync、プライマリー / セカンダリーが Primaryの場合
NTPServer		NTPサーバー	NTP同期がONの場合

リターン例

成功時

同期プロトコルがNTPの場合：
OK TimezoneLocation=Asia&TimezoneCity=Tokyo&CurrentTime=20250925101010&SyncProtocol=0&NTPSynchronization=true&NTPServer=192.168.0.1

'同期プロトコルがSoftwareSyncの場合：
OK TimezoneLocation=Asia&TimezoneCity=Tokyo&CurrentTime=20250925101010&SyncProtocol=1&DomainNo=0&OrderPs=0
&NTPSynchronization=true&NTPServer=192.168.0.1

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

	タイムゾーンの設定

コマンド	SetTimeZone

内容

タイムゾーンを設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
TimezoneLocation	○	タイムゾーン 地域	
TimezoneCity	○	タイムゾーン 都市	

リクエスト例

```
SetTimeZone TimezoneLocation=Africa&TimezoneCity=Abidjan
```

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

```
OK
```

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

```
NG Invalid parameter
```

※：NGは、以下の要因が考えられます
・パラメータ不正(Invalid parameter)

	日時情報の設定

コマンド	**SetDatetimeProperty**

内容

日時情報を設定するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
SyncProtocol	○	同期プロトコル(0:NTP 1:SoftwareSync)	
DomainNo		ドメイン番号(0 - 127)	同期プロトコルがSoftwareSyncの場合、設定必要
OrderPs		プライマリー / セカンダリー (0:Primary 1:Secondary)	同期プロトコルがSoftwareSyncの場合、設定必要
NTPSynchronization		NTP同期(true:ON false:OFF)	同期プロトコルがNTPの場合、設定必要 同期プロトコルがSoftwareSync、プライマリー / セカンダリーが Primaryの場合、設定必要
NTPServer		NTPサーバー	NTP同期がONの場合、設定必要
Date		日付(YYYY/MM/DD)	NTP同期がOFF以外の場合、設定内容を無視
Time		時刻(hh:mm:ss)	NTP同期がOFF以外の場合、設定内容を無視

リクエスト例

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

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

```
OK


※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります
```

失敗時

```
NG Invalid parameter

※：NGは、以下の要因が考えられます
  ・パラメータ不正(Invalid parameter)
```

	Webアカウント情報の変更

コマンド	**SetAccountProperty**

内容

Webアカウント情報を変更するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
CurrentUserName	○	現在のユーザー名	
CurrentPassword	○	現在のパスワード	
UserName		ユーザー名	ユーザー名とパスワードがすべて未設定の場合、エラー
Password		パスワード	

リクエスト例

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

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
- パラメータ不正(Invalid parameter)
- 認証エラー(Authentication failed)
- 既存アカウントと同じ(The same value is entered)

	システムの初期化

コマンド	ExecInitialize

内容

システムを初期化するためのコマンド

※処理に時間がかかる為応答まで最大20秒かかる場合があります
※本コマンド実行後、自動で再起動します

リクエスト

パラメータ名	必須	内容	備考
All	○	すべての設定(0:全初期化 1:部分初期化)	
Player		プレーヤー設定(true, false)	すべての設定が"1:部分初期化"の場合、最低１項目をtrueに設定が必要です 「All」/「Network」を設定する場合、IPアドレスが初期化される
Geometory		幾何学補正設定(true, false)	
Network		ネットワーク設定(true, false)	
Content		コンテンツデータ(true, false)	

リクエスト例

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

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
・パラメータ不正(Invalid parameter)
・認証エラー(Authentication failed)

	システム再起動

コマンド	Reboot

内容

システムを再起動するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

Reboot

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

なし

●Sample code for Setting Control（python）

設定制御を行うサンプルコードです。

使用するFMPのIPアドレス（ローカルIP）に併せて「DEFAULT_FMP_IP」「BROADCAST_IP」を変更して使用ください。

```
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 (環境に合わせて変更してください)
FMP_PORT = 65430  # 設定制御コマンド用のポート番号

# 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 (環境に合わせて変更してください)

# --- 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 ---
# 必要に応じて、設定コマンド固有のグローバル変数を追加

# --- 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 プレイヤー再生状況の取得 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 プレイヤー情報の取得 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 プレイヤー情報の設定 SetPlayerProperty
    ※処理に時間がかかる為応答まで最大30秒かかる場合があります"""
    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 を必須にする
    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}") # VideoSignalを追加
params.append(f"AudioOutput={audio_output}")

# OutputModeが4の場合のみIPアドレス設定が必要（必須）
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 オーディオ情報の取得 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 オーディオ情報の設定 SetAudioProperty"""
    print("\n--- SetAudioProperty ---")

    hdmi_volume = ""
    while not hdmi_volume.isdigit() or not (0 <= int(hdmi_volume) <= 100): # HdmiVolumeを必須にする
        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を必須にする
    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を必須にする
    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): ") # 必須ではない

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

if not params: # 何も設定しない場合はエラーとする
    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 高度な設定情報の取得 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 高度な設定情報の設定 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 NDIストリーム一覧の取得 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 NDIストリーミング再生の開始 PlayStream
    ※処理に時間がかかる為応答まで最大10秒かかる場合があります"""
    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 ストリーミング再生の停止 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 自動再生ストリーム情報の取得 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 自動再生ストリーム情報の設定 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を必須にする
        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 プレイリスト同期情報の取得 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 プレイリスト同期情報の設定 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("PlayContolSync (true/false, required): ")
        if play_control_sync not in ['true', 'false']:
            print("Invalid value for PlayContolSync. Please enter true or false.")

    params = []
    params.append(f"PlaylistSync={playlist_sync}")
    params.append(f"PlayContolSync={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 表示言語の取得 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 表示言語の設定 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 ホスト名の取得 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 ホスト名の設定 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 POWER LED情報の取得 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 POWER LED情報の設定 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 ネットワーク情報の取得 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 ネットワーク情報の設定 SetNetworkProperty
    ※処理に時間がかかる為応答まで最大20秒かかる場合があります"""
    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 日時情報の取得 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 タイムゾーンの設定 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 日時情報の設定 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': # SoftwareSyncの場合
        domain_no = ""
```

```
while not domain_no.isdigit() or not (0 <= int(domain_no) <= 127): # DomainNoを必須にする
    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を必須にする
    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': # NTPの場合
    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 Webアカウント情報の変更 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 システムの初期化 ExecInitialize
    ※処理に時間がかかる為応答まで最大20秒かかる場合があります。本コマンド実行後、自動で再起動します。"""
    print("\n--- ExecInitialize ---")
    all_init = ""
    while all_init not in ['0', '1']:
        all_init = input("All (0:全初期化, 1:部分初期化, 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': # 部分初期化の場合、最低1項目がtrueである必要
        player = input("Player (true/false, optional): ")
        geometry = input("Geometory (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"Geometory={geometry}")
        if network: params.append(f"Network={network}")
        if content: params.append(f"Content={content}")

        # 部分初期化の場合、最低1項目がtrueである必要
        if not (player == 'true' or geometry == 'true' or network == 'true' or content == 'true'):
            print("For partial initialization, at least one item (Player, Geometory, 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 システム再起動 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 ログ出力レベルの取得 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 ログ出力レベルの設定 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  # 現在のターゲット IP
    current_send_mode = '1'  # デフォルトはシングル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.")
```

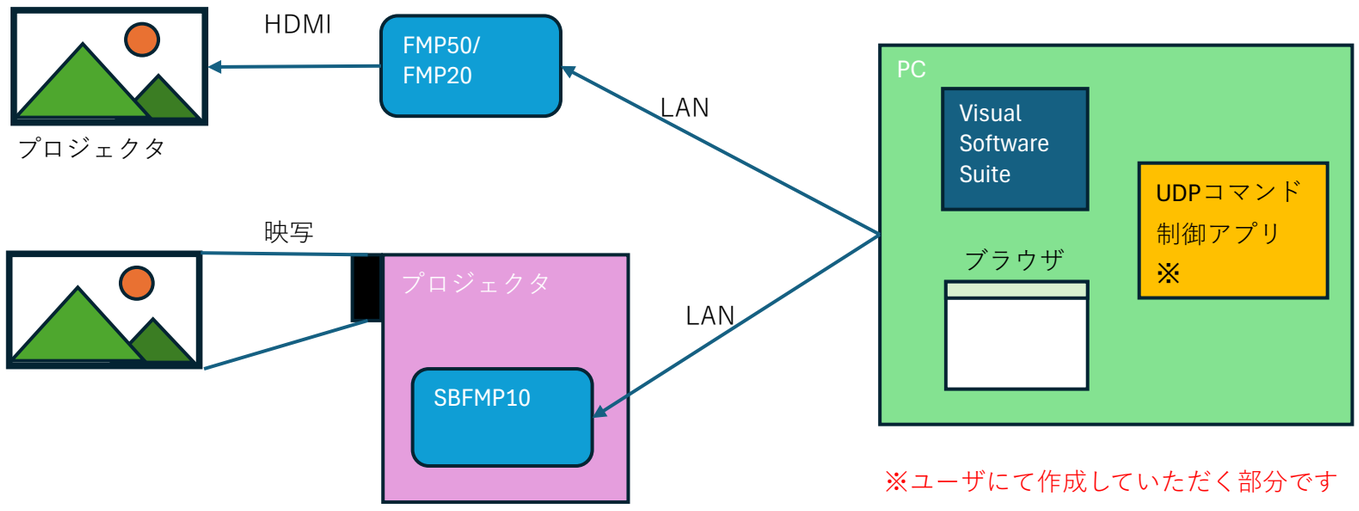
■UDPコマンド使用方法(再生制御)

●概要

メディアプロセッサ（FMP）におけるTimeline機能によるメディアの再生操作をUDP通信を用いたコマンドにておこなえます。
メディアの再生操作はプレイリスト（※）単位でおこないます。
メディアプロセッサ（FMP）にはプレイリストを最大９９まで登録可能で、プレイリストを切り替えてのメディアの再生操作がおこなえます。
また、複数のメディアプロセッサ（FMP）によるメディアの同期再生も可能です。

※：プレイリストには個別のメディア（動画や静止画）を複数含めることができ、１つのメディアとして再生操作をおこなえます。

●構成図

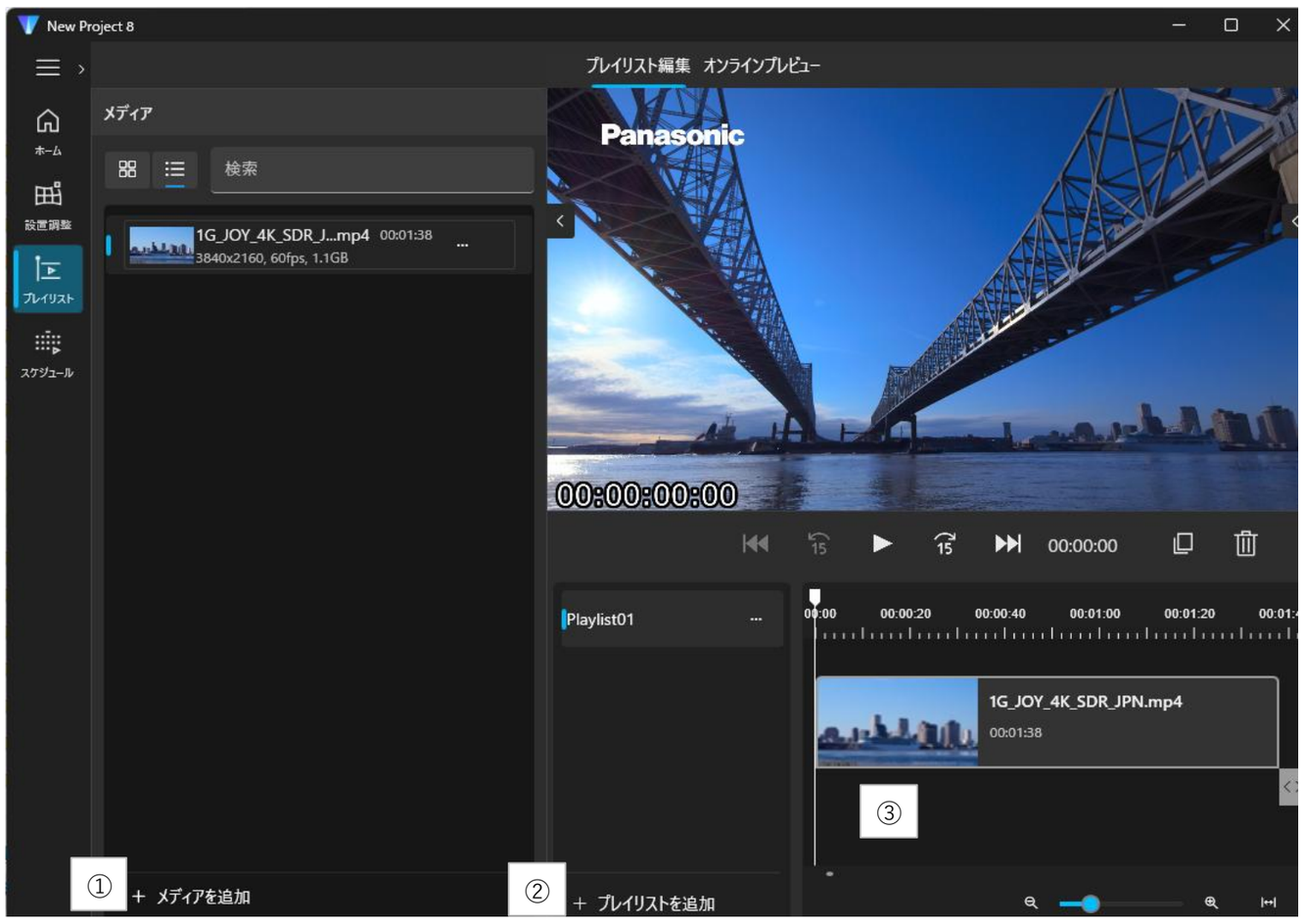


※ユーザにて作成していただく部分です

●VSSを用いたプレイリストの作成および、メディアプロセッサへのプレイリスト送信

※：詳細はVisualSoftwareSuite（VSS）のマニュアルを参照ください。

１．プレイリストの作成は下記のプレイリスト編集画面にて行います。
①プレイリストに登録するメディアの追加をする。
②プレイリストを作成する。
③①で追加したメディアを②で作成したプレイリストに登録する。

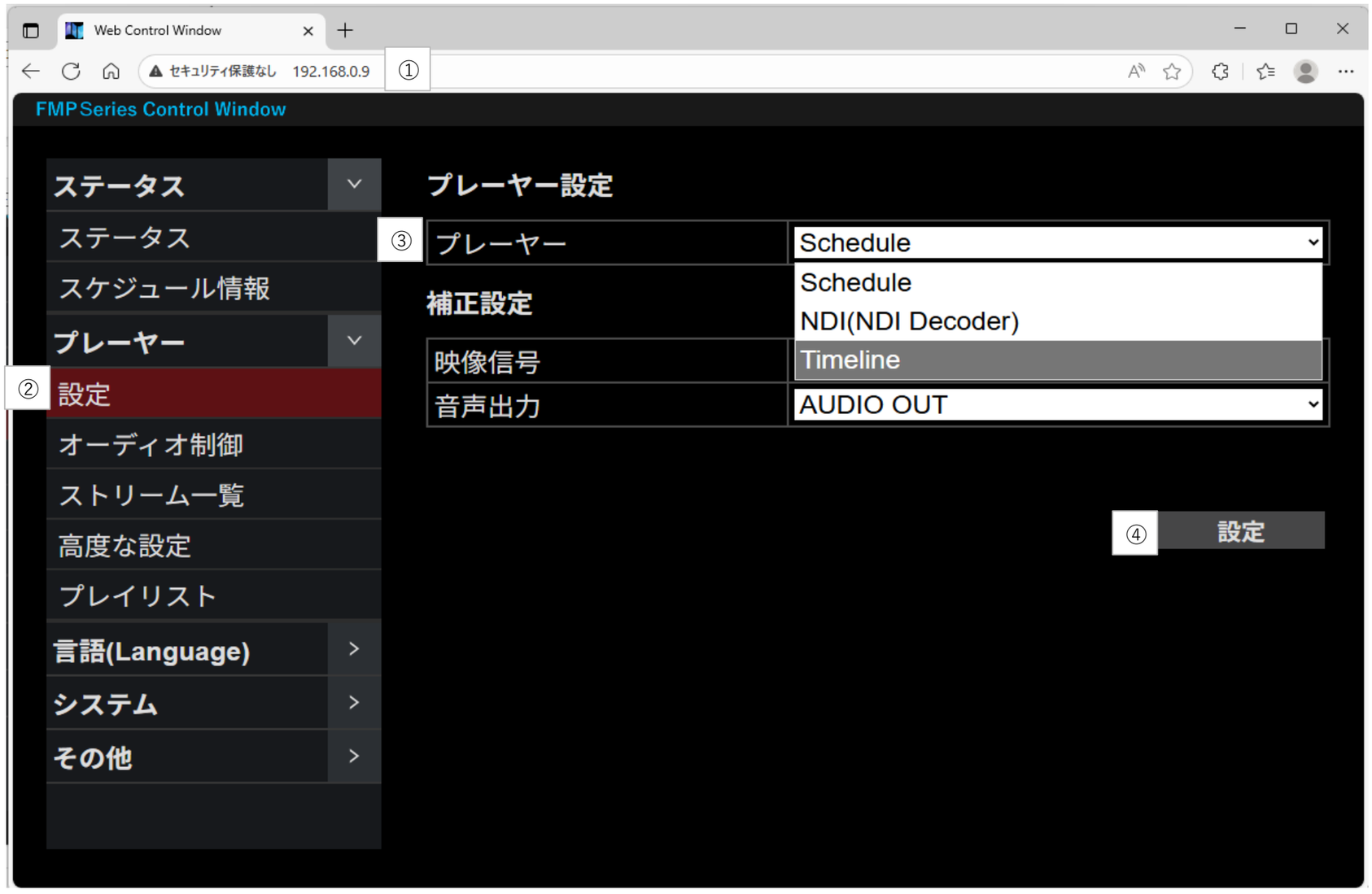


2．作成したプレイリストはオンラインプレビュー画面にてメディアプロセッサ（FMP）への送信を行います。

①プレイリストを送るグループを追加する。
②プレイリストを送るメディアプロセッサ（FMP）の追加をする。
③プレイリスト編集で作成したプレイリストを追加する。
④③実施後にデータ転送のポップアップが出るので、OKを押下してメディアプロセッサ（FMP）へのプレイリスト送信をする。
⑤ポップアップが出ない場合は、プレイリスト更新押下により④のポップアップが出る。（以降は④と同じ動作をおこなう）

●WEB（FMP Series Control Window）にて、プレーヤー：Timelineの起動の手順

①WEBブラウザにて、該当のメディアプロセッサ（FMP）のIPアドレスにアクセスする。
②左側メニューから「プレーヤー」→「設定」を押下し、下記画面に遷移する。
③プレーヤー設定の「プレーヤー」から「Timeline」を選択する。
④設定ボタンを押下する。
⑤数秒後、メディアプロセッサ（FMP）でプレーヤー「Timeline」が’起動する。

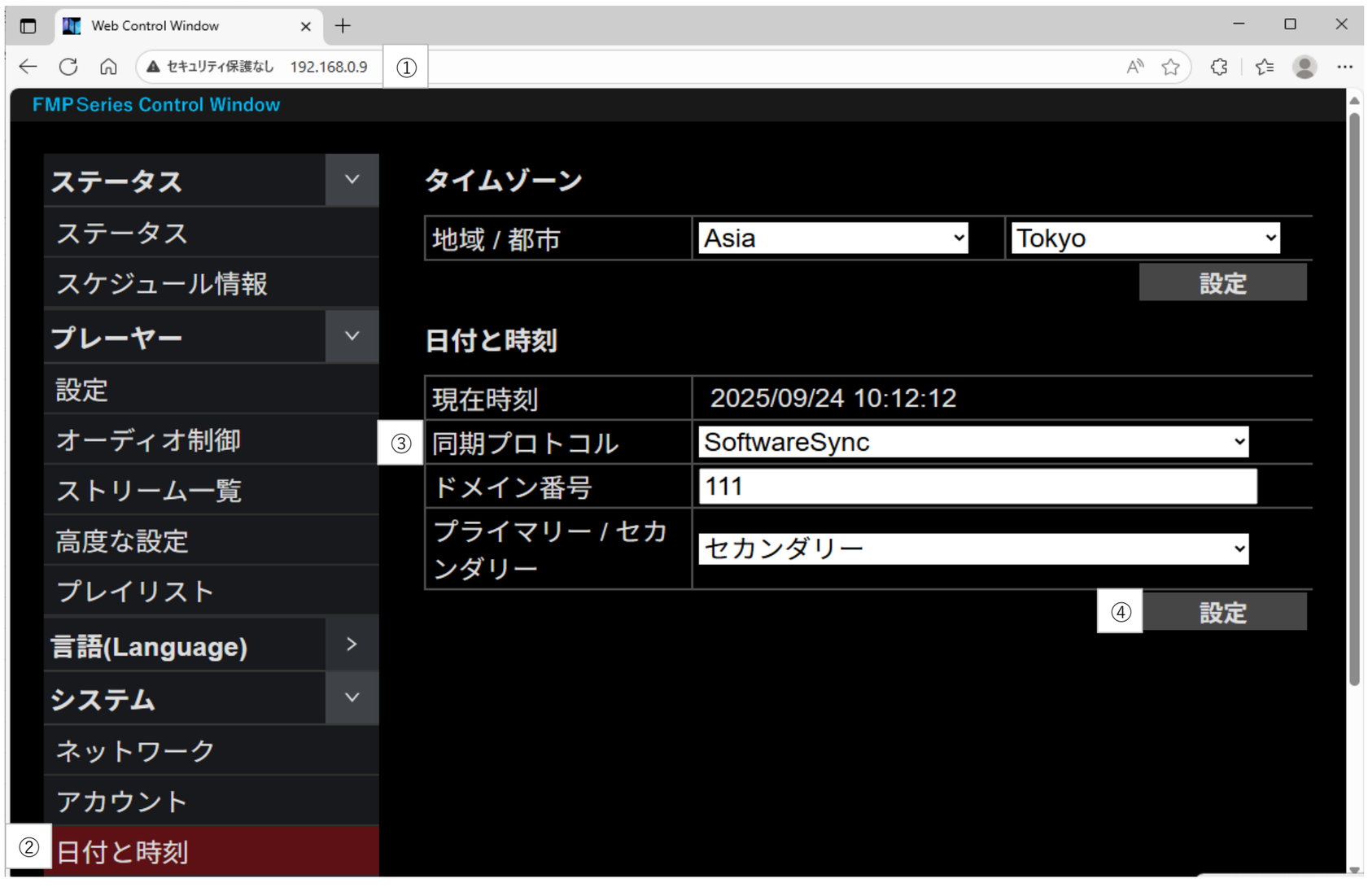


●WEB（FMP Series Control Window）にて、メディア同期再生をおこなうための手順

※：ここでは2台のメディアプロセッサによるメディア同期再生のための設定について説明します。詳細はマニュアルを参照ください。

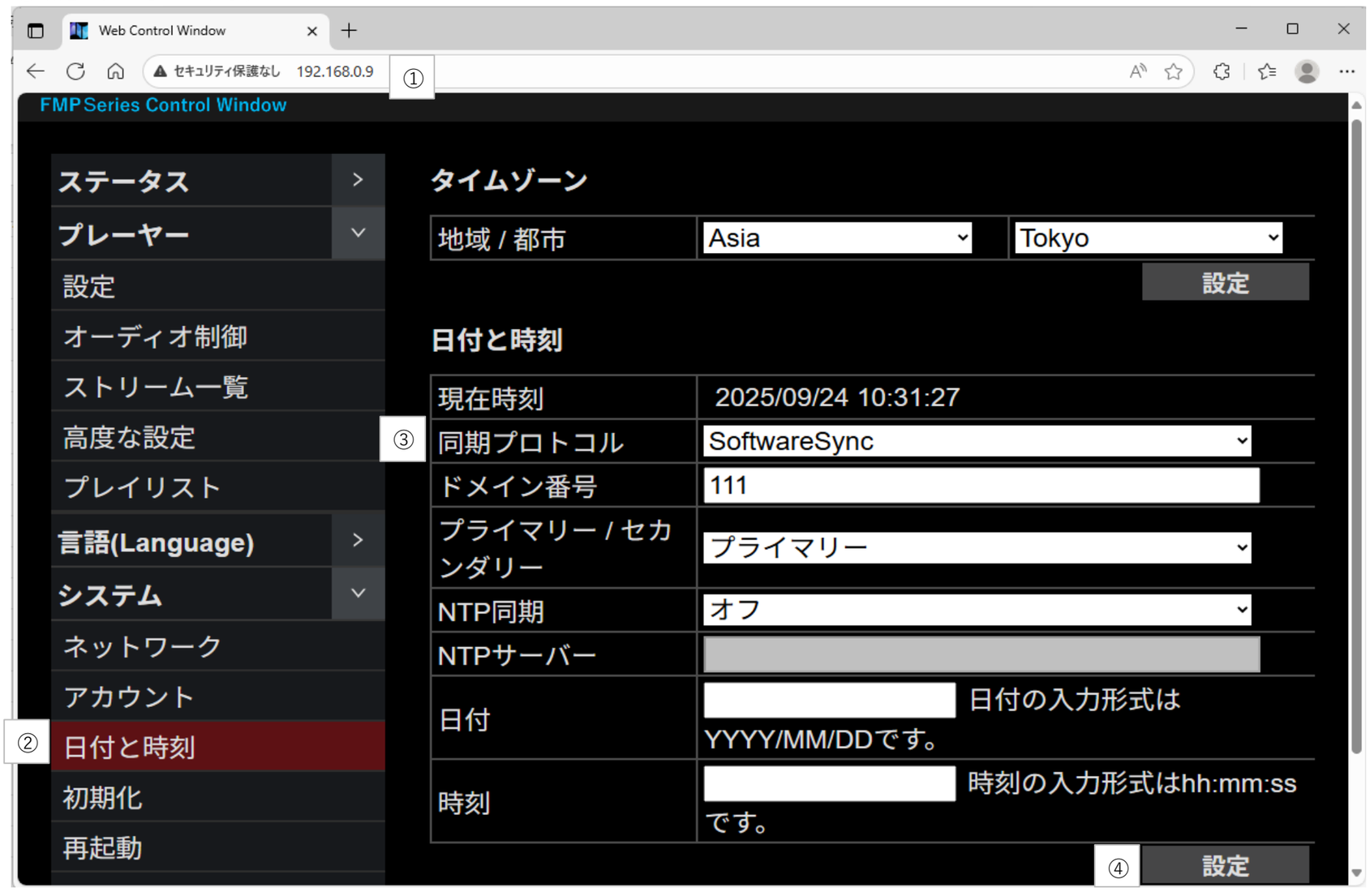
・セカンダリー側

①WEBブラウザにて、セカンダリーにするメディアプロセッサ（FMP）のIPアドレスにアクセスする。
②左側メニューから「システム」→「日付と時刻」を押下し、下記画面に遷移する。
③日付と時刻の「同期プロトコル」から「SoftwareSync」を選択し、
　「ドメイン番号」に任意の値（ここでは「111」）に設定し、
　「プライマリー／セカンダリー」から「セカンダリー」を選択する。
④設定ボタンを押下する。



・プライマリー側

①WEBブラウザにて、プライマリーにするメディアプロセッサ（FMP）のIPアドレスにアクセスする。
②左側メニューから「システム」→「日付と時刻」を押下し、下記画面に遷移する。
③日付と時刻の「同期プロトコル」から「SoftwareSync」を選択し、
　「ドメイン番号」に任意の値（ここでは「111」）に設定し、
　「プライマリー／セカンダリー」から「プライマリー」を選択する。
④設定ボタンを押下する。



●UDPコマンド仕様

FMP側接続条件

プレーヤー：Timelineが起動し、以下のソケットで待ち受けしている。

アドレス：**FMPに設定したアドレス**

待ち受けポート番号：**65432**

接続元アドレス：**INADDR_ANY**

最大送受信サイズ：**65536byte**

操作系コマンド一覧

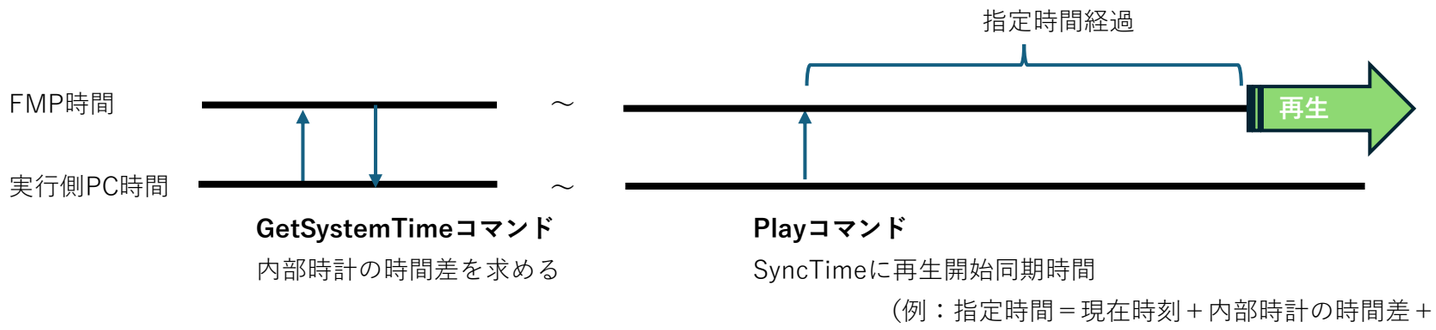
コマンド	説明
GetSystemTime	**UDPコマンド実行側とFMP側の時間調整**
Play	**プレイリスト再生**
Pause	**プレイリスト再生の停止** 指定した位置で再生停止する
Stop	**プレイリスト再生の停止** コマンド受信時点で再生停止する （停止位置を指定しなくてよい）
Seek	**プレイリスト再生のシーク**
Skip	**プレイリスト内のメディアスキップ**
SelectPlayList	**プレイリスト選択**
Loop	**プレイリストのループ設定**
DisconnectReq	**モジュールの再起動**
ResetPlaylist	**プレイリストの再読み込み**
SetSyncTime	**FMP側の時間調整の設定** 初期値は3秒で設定されている

情報系コマンド一覧

コマンド	説明
GetPlaylistInfo	**プレイリスト情報の取得**
GetEachPlaylistInfo	**プレイリスト内のメディア情報の取得**
GetPlayingInfo	**プレイリスト再生状況の取得**

プレイリストの再生までの手順（再生開始同期時間を指定する場合）

【処理イメージ】



GetSystemTimeコマンド
内部時計の時間差を求める

Playコマンド
SyncTimeに再生開始同期時間
（例：指定時間＝現在時刻＋内部時計の時間差＋N秒）指定

【手順】

①GetSystemTime（UDPコマンド実行側とFMP側の時間調整）を実行
②UDPコマンド実行側PCとFMP側の内部時計の時間差を求め、
　以降のUDPコマンドのパラメータ「再生開始同期時間」に加算すること
③SelectPlayList（プレイリスト選択）を実行
④Loop（プレイリストのループ設定）を実行
⑤Play（プレイリスト再生）を実行

【コマンド例】

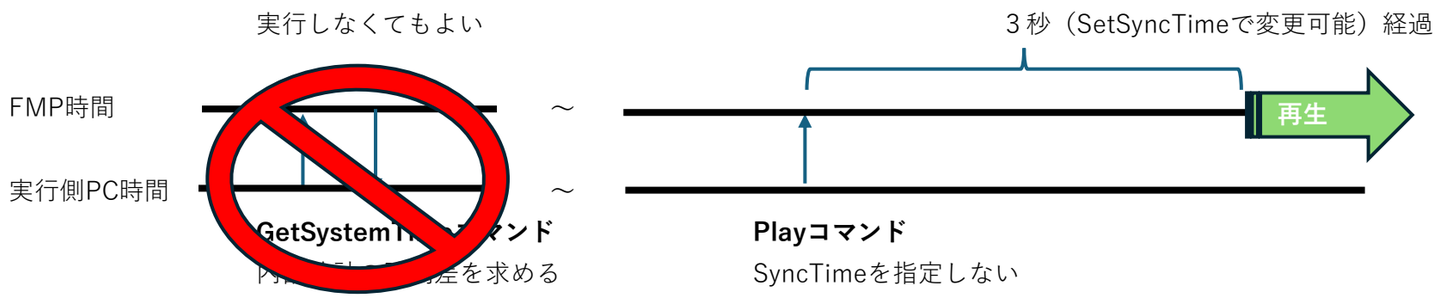
GetSystemTime CurrentTimeVss=1715237197598000
ー

SelectPlayList PlayListName=Playlist01
Loop EnableLoop=0
Play SyncTime=1727684152203481&PlayPosition=0

以降は、Pause、Stop、Seek、Skipのコマンドでプレイリストの再生停止等をおこなう。

プレイリストの再生までの手順（再生開始同期時間を指定しない場合）

【処理イメージ】



【手順】

①SelectPlayList（プレイリスト選択）を実行
②Loop（プレイリストのループ設定）を実行
③Play（プレイリスト再生）を実行　※

【コマンド例】

SelectPlayList PlayListName=Playlist01
Loop EnableLoop=0
Play

以降は、Stop、Seek※、Skip※のコマンドでプレイリストの再生停止等をおこなう。

※：FMPがコマンド受信から3秒後（SetSyncTimeで変更可能）に再生等を実施する

プレイリストの再生状況の確認

【手順】

①プレイリスト再生状態にする
②GetPlayingInfo（プレイリスト再生状況の取得）を任意のタイミングで実行
③GetPlayingInfo（プレイリスト再生状況の取得）のリターンのCurrentPosition（プレイリストの現在位置）の増加で
　プレイリストの再生状況を判断する

プレイリストの更新後のプレイリスト再読み込み

【手順】

①VSSからFMP内のプレイリストの更新を実行
②DisconnectReq（モジュール再起動）または、ResetPlaylist（プレイリストの再読み込み）を実行
③GetPlaylistInfo（プレイリスト情報の取得）、GetEachPlaylistInfo（プレイリスト内のメディア情報の取得）を実行し、
　プレイリストが更新されたことを確認
④「プレイリストの再生までの手順」を実行

	UDPコマンド実行側とFMP側の時間調整

コマンド	**GetSystemTime**

内容

UDPコマンド実行側PCとFMP側の内部時計の時間差を求めるためのコマンド

なお、時間差はマイクロ秒単位で求め、この値を以降のUDPコマンドのパラメータ「再生開始同期時間」に加算してください

リクエスト

パラメータ名	必須	内容	備考
CurrentTimeVss	○	UDPコマンド実行側時間（Linux時間＋μ秒）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値

リクエスト例

GetSystemTime CurrentTimeVss=1715237197598000

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります
※２：1715237197598000＝2024/05/09 06:46:37.598000

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
CurrentTimeVss	○	リクエスト時に設定されたUDPコマンド実行側時間	
CurrentTimeFmp	○	FMP側時間（Linux時間＋μ秒）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値

リターン例

成功時

OK CurrentTimeVss=1715237197598000&CurrentTimeFmp=1715237196004200

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります
※２：1715237197598000＝2024/05/09 06:46:37.598000
　　1715237196004200＝2024/05/09 06:46:36.004200

失敗時

なし

	プレイリスト再生

コマンド	**Play**

内容

FMPに登録されているプレイリストにおいて、指定されたプレイリストの再生をおこなうためのコマンド

リクエスト

パラメータ名	必須	内容	備考
SyncTime		再生開始同期時間（Linux時間＋μ秒）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値 なおFMP内部処理のため、１秒以上の未来の時間を指定すること また、GetSystemTimeでの時間差も含めること
PlayPosition		プレイリストの再生位置（先頭からの経過時間）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値

リクエスト例

再生開始同期時間指定の場合：Play SyncTime=1727684152203481&PlayPosition=0

再生開始同期時間指定なしの場合：Play PlayPosition=0　または　Play

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります
※２：プレイリストの再生位置を０以外にするとプレイリストの途中から再生開始することができます
※３：再生開始同期時間指定無しの場合は、「SyncTime」および「PlayPosition」のパラメータを省略できます
　　その場合、FMPが本コマンド受信から3秒後（SetSyncTimeで変更可能）にプレイリストの先頭から再生します

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	
NG要因		NGの場合、その要因を通知	

リターン例

成功時

OK

失敗時

NG Playlist not selected

※：NGは、以下の要因が考えられます
　・プレイリスト指定がされていない→Playlist not selected
　・プレイリストの再生位置がプレイリストの範囲外を指している→Out of range PlayPosition:（指定した再生位置）

	プレイリスト再生の停止

コマンド	**Pause**

内容

指定するプレイリストの停止位置でプレイリストの再生を停止をおこなうためのコマンド

リクエスト

パラメータ名	必須	内容	備考
StopPosition	○	プレイリストの停止位置（先頭からの経過時間）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値

リクエスト例

Pause StopPosition=3800490

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

失敗時

NG

※：NGは、以下の要因が考えられます
　・プレイリストの停止位置がプレイリストの範囲外を指している

	プレイリスト再生の停止

コマンド	Stop

内容

コマンド受信時点での位置でプレイリストの再生停止をおこなうためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

Stop

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	
CurrentPosition	○	プレイリストの停止位置（先頭からの経過時間）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値 この値をプレイリスト再生の再生位置に設定すると、 停止した位置からプレイリスト再生する

リターン例

成功時

OK CurrentPosition=10000000

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります
※２：10000000マイクロ秒＝10秒

失敗時

NG CurrentPosition=-1

※：NGは、以下の要因が考えられます
　・プレイリスト再生していない
　・FMP内の再生停止処理中

	プレイリスト再生のシーク

コマンド	**Seek**

内容

プレイリスト再生中に指定するプレイリストの再生位置からプレイリストの再生をおこなうためのコマンド

なお、プレイリスト再生停止中の場合には指定するプレイリストの再生位置に移動するだけになります

リクエスト

パラメータ名	必須	内容	備考
SyncTime		再生開始同期時間（Linux時間＋μ秒）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値 なおFMP内部処理のため、１秒以上の未来の時間を指定すること また、GetSystemTimeでの時間差も含めること
PlayPosition	○	プレイリストの再生位置（先頭からの経過時間）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値

リクエスト例

再生開始同期時間指定の場合：Seek SyncTime=1727669330084099&PlayPosition=0

再生開始同期時間指定なしの場合：Seek PlayPosition=0

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります
※２：プレイリストの再生位置を０以外にするとプレイリストの途中から再生または移動することができます
※３：再生開始同期時間指定無しの場合は、「SyncTime」のパラメータを省略できます
　　その場合、FMPが本コマンド受信から3秒後（SetSyncTimeで変更可能）に再生します

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	
NG要因		NGの場合、その要因を通知	

リターン例

成功時

OK

失敗時

NG Playlist not selected

※：NGは、以下の要因が考えられます
・プレイリスト指定がされていない→Playlist not selected
・別コマンド実行中→Other command running
・プレイリストの再生位置がプレイリストの範囲外を指している→Out of range PlayPosition:（指定した再生位置）

	プレイリスト内のメディアスキップ

コマンド	Skip

内容

プレイリスト再生中のメディアから次のメディアにスキップをおこなうためのコマンド

なお、プレイリスト停止中の場合はスキップはおこなわれません
また、プレイリストに登録されているメディアが1つしかない場合はスキップはおこなわれず、再生停止します
（ループ設定ON時は先頭に戻ります）

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

Skip

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	
NG要因		NGの場合、その要因を通知	

リターン例

成功時

OK

失敗時

NG Playlist not selected

※：NGは、以下の要因が考えられます
・プレイリスト指定がされていない→Playlist not selected
・プレイリスト再生していない→Playlist not playing
・別コマンド実行中→Other command running

	プレイリスト選択

コマンド	SelectPlayList

内容

FMPに登録されているプレイリストにおいて、再生するプレイリストを選択するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
PlayListName	○	選択するプレイリスト名	

リクエスト例

SelectPlayList PlayListName=Playlist01

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

失敗時

NG

※：NGは、以下の要因が考えられます
　・該当プレイリストがFMP内に存在していない

	プレイリストのループ設定

コマンド	Loop

内容

プレイリスト再生をプレイリスト内でループするかの選択をするためのコマンド

初期値はループ無（終端で停止）で設定されています

リクエスト

パラメータ名	必須	内容	備考
EnableLoop	○	ループ設定（0:ループ無 1:ループ有）	ループ無（終端で停止）、ループ有（再度先頭から再生）

リクエスト例

Loop EnableLoop=0

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ

リターン例

成功時

OK

失敗時

なし

	モジュールの再起動

コマンド	DisconnectReq

内容

FMP内で動作しているモジュールを再起動するためのコマンド

※モジュール再起動は「プレイリストの入れ替え」時におこなってください

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

DisconnectReq

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ

リターン例

成功時

OK

失敗時

なし

	プレイリストの再読み込み

コマンド	ResetPlaylist

内容

FMP内で動作しているモジュールに取り込んだプレイリスト情報を更新するためのコマンド

※本コマンドは「プレイリストの入れ替え」時におこなってください
　また、プレイリスト再生中に本コマンドを実行した場合、プレイリスト再生の停止をします
　以降はプレイリスト選択からおこなってください

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

ResetPlaylist

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ

リターン例

成功時

OK

失敗時

なし

	FMP側の時間調整の設定

コマンド	SetSyncTime

内容

再生開始同期時間を指定しない場合における、FMP内でのコマンド実行までの時間調整値を変更するためのコマンド

※初期値は3秒で設定されています
　また、FMPの再起動、Timelineモジュールの再起動時には本調整値の値は３秒に戻ります

リクエスト

パラメータ名	必須	内容	備考
Time	○	時間調整値	秒数を整数で指定 ３秒以下の場合は、同期再生等がおこなえない可能性あり

リクエスト例

SetSyncTime Time=5

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ

リターン例

成功時

OK

失敗時

なし

	プレイリスト情報の取得

コマンド	GetPlaylistInfo

内容

FMPに登録されているプレイリスト名一覧を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
-d		デバッグオプション	VSSで使用しているID部分を付けて通知する

リクエスト例

```
GetPlaylistInfo

GetPlaylistInfo -d
```

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）になります

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	
Playlist1		プレイリスト名	NGまたは未登録時は設定しない
Playlist2		プレイリスト名	NGまたは未登録時は設定しない
Playlist3		プレイリスト名	NGまたは未登録時は設定しない
Playlist4		プレイリスト名	NGまたは未登録時は設定しない

リターン例

成功時

```
OK Playlist1=Weekday&Playlist2=Weekend

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります
※２：上記例ではプレイリストが2つしか登録されていないので「Playlist2」までしかありません

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

※３：デバッグオプション指定時は上記のようにID部分を付けて通知します
```

失敗時

```
NG

※：NGは、以下の要因が考えられます
  ・プレイリストが１つも登録されていない
```

	プレイリスト内のメディア情報の取得

コマンド	GetEachPlaylistInfo

内容

FMPに登録されているプレイリスト内のメディア情報（メディア名、再生時間）を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
Playlist	○	メディア情報を取得するプレイリスト名	VSSで使用しているID部分は必要ない

リクエスト例

GetEachPlaylistInfo Playlist=Weekday

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	
Playlist	○	リクエスト時に設定されたプレイリスト名	
Medianum	○	プレイリストに含まれるメディアの数	
Media1		1つ目のメディア名	
Duration1		1つ目のメディアの再生時間	単位は「秒」
Media2		2つ目のメディア名	
Duration2		2つ目のメディアの再生時間	単位は「秒」
・・・		・・・	
MediaN		Nつ目のメディア名	
DurationN		Nつ目のメディアの再生時間	単位は「秒」

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります
※２：上記例ではプレイリスト内にメディアが3つしか登録されていないので「Media3」、「Duration3」までしかありません
※３：プレイリスト内に多数のメディアが登録されていると最大送受信サイズ：1024byteを超える場合があるが、その場合は1024byteまでを通知します

失敗時

NG Playlist=PLAYLIST1&Medianum=0

※：NGは、以下の要因が考えられます
　・該当プレイリスト存在していない

	プレイリスト再生状況の取得

コマンド	GetPlayingInfo

内容

FMPでプレイリスト再生されている状況（プレイリストの現在位置で判別）を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

GetPlayingInfo

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	
Playlist	○	プレイリスト名	
CurrentPosition	○	プレイリストの現在位置	再生停止時は「STOP」それ以外のNGは空白
Loop	○	ループ設定（0:ループ無 1:ループ有）	ループ無（終端で停止）、ループ有（再度先頭から再生）

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

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

※２：プレイリスト再生停止時（Pause、Stop実行後）はプレイリストの現在位置を「STOP」で通知します

失敗時

NG Playlist=&CurrentPosition=&Loop=0

※：NGは、以下の要因が考えられます
・プレイリスト指定がされていない

●Sample code for Playback Control（python）

再生制御（Timeline）を行うサンプルコードです。
使用するFMPのIPアドレス（ローカルIP）に併せて「DEFAULT_FMP_IP」「BROADCAST_IP」を変更して使用ください。

```
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 # FMPとの時間差（マイクロ秒）
marge_time_us = 3000000 # FMPの処理時間猶予 (3秒 = 3,000,000マイクロ秒) - 3秒以上推奨
PLAY_POSITION = 0 # 再生開始位置 (us単位)
PLAYLIST_NAME = "Playlist01" # デフォルトのプレイリスト名
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):
    """
    GetSystemTimeコマンドを実行し、FMPとの時間差(diff_time_us)を更新します。
    ブロードキャストの場合、最初に応答したFMPの時刻を使用します。
    """
    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':
        # PLAY_POSITIONはStop/Pause時に更新されていることを想定
        print(f"Using current PLAY_POSITION: {PLAY_POSITION} us")
    else:
        print("Invalid choice. Exiting play settings.")
        return

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

    # SyncTimeの指定有無をユーザーに確認
    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 = 現在時刻(PC) + マージタイム  - 時間差
        sync_time_val = int(time.time() * 1000000) + marge_time_us - diff_time_us
        command_params.insert(0, f"SyncTime={sync_time_val}") # SyncTimeを先頭に追加

    # 最終的なコマンド文字列の構築
    # Play Command: "Play SyncTime=...&PlayPosition=..." or "Play PlayPosition=..."
    final_command = "Play " + "&".join(command_params) # コマンド名と最初のパラメータはスペースで区切る
    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 # デフォルト値 (10秒)
            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 # デフォルト値 (10秒)
    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:
    # ここを修正: int() に変換する前に文字列の前後から空白文字を除去する
    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 # 停止状態なので、ポジションは0にリセット
        return
    else:
        try:
            pause_position = int(current_pos_str_trimmed) # strip() 適用後の文字列を使用
            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 # デフォルト値 (10秒)
else:
    print("CurrentPosition not found in GetPlayingInfo response. Using default pause position (10,000,000 us).")
    pause_position = 10000000 # デフォルト値 (10秒)

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 # Pauseした位置を次のPlay/Seekのために保存
else:
    print("Failed to send Pause command.")


def handle_stop(send_mode, target_ip):
    global PLAY_POSITION
    command = "Stop" # Stopコマンドにはパラメータなし
    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:
                # まず前後の空白文字をトリム
                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 # 完全に停止したのでポジションは0に
                else:
                    # さらに数字以外の文字を全て除去
                    # これにより、見えない制御文字やUnicode空白文字などを確実に排除
                    current_pos_str_cleaned = re.sub(r'[^0-9]', '', current_pos_str_trimmed)

                    try:
                        PLAY_POSITION = int(current_pos_str_cleaned) # クリーンアップ後の文字列を使用
                        print(f"Stop command sent successfully. Stopped at position: {PLAY_POSITION} us")
                    except ValueError:
                        # エラーメッセージをより詳細に表示
                        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 # パース失敗時はリセット
            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 # 失敗時はリセット
    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}"] # PlayPositionを最初に配置

# SyncTimeの指定有無をユーザーに確認
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 = 現在時刻(PC) + マージタイム  - 時間差
        sync_time_val = int(time.time() * 1000000) + marge_time_us - diff_time_us
        command_params.insert(0, f"SyncTime={sync_time_val}") # SyncTimeを先頭に追加

    # 最終的なコマンド文字列の構築
    # 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 # Seekした位置を保存
    else:
        print(f"Failed to send Seek command to position: {seek_position} us")

def handle_skip(send_mode, target_ip):
    global PLAY_POSITION

    # SkipコマンドにはSyncTimeパラメータがないため、シンプルにコマンドを送信
    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 # Skip後、通常は先頭に戻るか次のメディアの先頭
    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 # 現在のターゲットIP
    current_send_mode = '1' # デフォルトはシングル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':
```

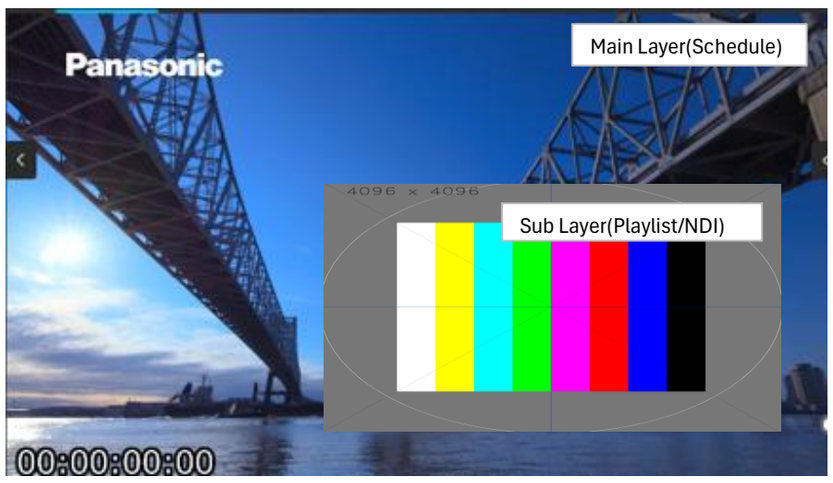
```
            handle_play(current_send_mode, current_target_ip)
        elif choice == '5':
            handle_pause(current_send_mode, current_target_ip)
        elif choice == '6':
            handle_stop(current_send_mode, current_target_ip)
        elif choice == '7':
            handle_seek(current_send_mode, 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.")
```

■UDPコマンド使用方法(PinP再生制御)

●概要

メディアプロセッサ（FMP）によるスケジュール再生中に、PinP（Picture in Picture）による割り込み再生をUDPコマンド通信でおこなえます。
割り込み再生出来るメディアは"プレイリスト/NDI"となります。

※：スケジュールやプレイリストの登録についての詳細はVisualSoftwareSuite（VSS）のマニュアルを参照ください。

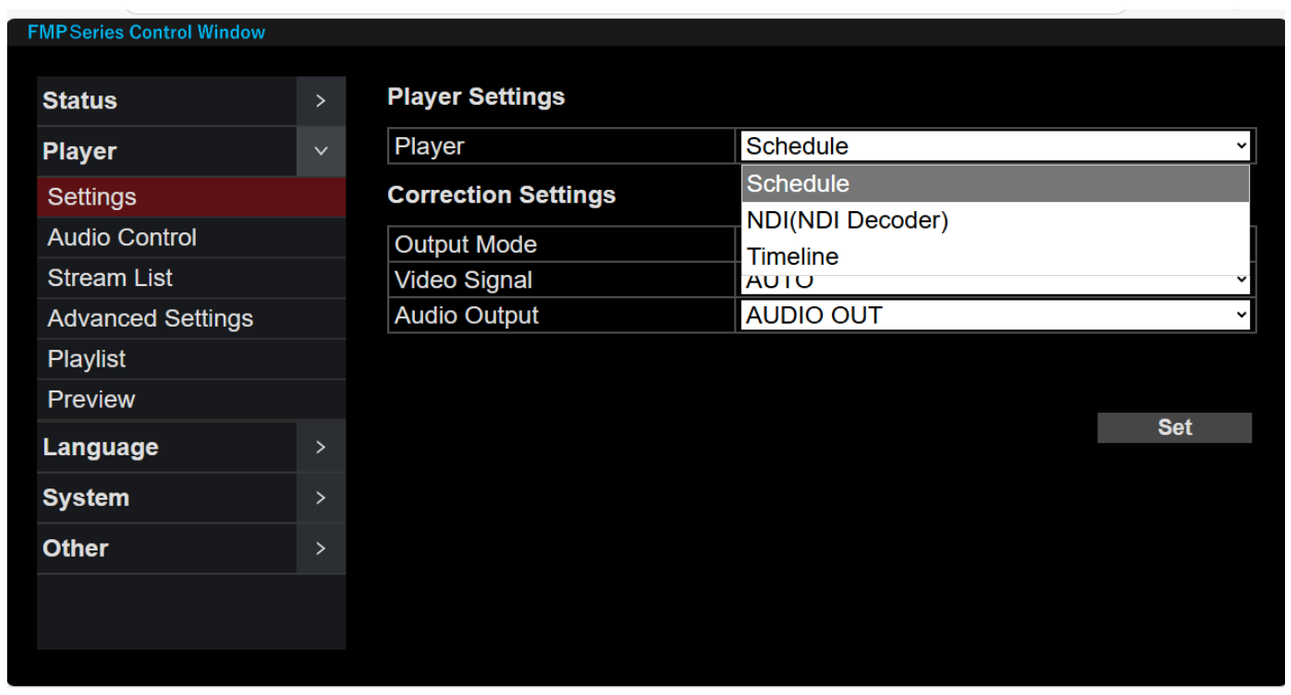


Main Layer : Schedule再生の映像です

Sub Layer : PinP(Playlist/NDI)の割り込み再生映像です

●WEB（FMP Series Control Window）設定

PinPによる割り込み再生を使用するためには、プレーヤ設定が「**Schedule**」の必要があります



●PinP/割り込み時の制約・条件

プレイリスト/NDI共通：

映像出力：メイン、割り込み側ともに30fps（60pコンテンツの再生は可能、30fpsに間引かれる）

プレイリスト:

HAP：メイン側、割り込み側共に非対応

メイン側、割り込み側共に最大150Mbp以内

NDI:

解像度：Full HD （4K非対応）

フレームレート：30/25

bitrate：最大100Mbps以内を推奨

上記を超える解像度、フレームレート、bitrateの場合、受像可能ですが、カクツキ、停止、同期ずれが発生する場合があります。

●UDPコマンド仕様

FMP側接続条件

プレーヤー：Scheduleが起動し、以下のソケットで待ち受けしている。

アドレス：**FMPに設定したアドレス**

待ち受けポート番号：**65432**

接続元アドレス：**INADDR_ANY**

最大送受信サイズ：**65536byte**

操作系コマンド一覧

コマンド	説明
pinpPlay	**割り込み再生(PinP)の開始**
pinpStop	**割り込み再生(PinP)の停止**
GetSystemTime	**UDPコマンド実行側とFMP側の時間調整**

情報系コマンド一覧

コマンド	説明
GetStreamList	**NDIストリーム一覧の取得**

	割り込み再生（PinP）開始

コマンド	**pinpPlay**

内容

PinP再生を開始するコマンド
プレーヤ設定がスケジュールの場合のみ動作します

リクエスト

パラメータ名	必須	内容	備考
PlayerType	○	割り込みするプレーヤータイプ(Timeline, NDI)	
Source	○	出力映像選択：PlayerType=Timeline時(プレイリスト名), NDI時(NDIソース名)	
Scale		出力映像サイズ[%](0 - 100)	省略時は[100]
PositionX		出力映像表示位置[X座標](0 - 100)	省略時は[0]
PositionY		出力映像表示位置[Y座標](0 - 100)	省略時は[0]
Audio		出力音声選択(Main, Sub)	出力音声を選択する。Main:Schedule側音声, Sub:PinP側音声 省略時は[Sub]
Loop		ループ設定(ON, OFF)	PlayerType=Timeline時のみ。省略時は[ON]。 [OFF]の場合はプレイリスト再生が1回終了したらPinP再生を終了する
Wait		再生開始までの待ち時間[Sec](0 - 30)	内部でデコード処理実施までに必要な時間を設定。設定時は3秒以上を設定すること。省略時は[0]で、内部で準備が出来たら再生開始。複数台で同期をとる場合などに十分な待ち時間を設定する SyncTimeが設定されている場合はSyncTimeが優先される。
SyncTime		再生開始同期時間(Linux時間＋μ秒)	Linux時間形式(小数点以下のμ秒を含む)を1000000倍した値なおFMP内部処理のため、3秒以上の未来の時間を指定すること。また、GetSystemTimeでの時間差も含めること "Wait"設定で正確な同期がとれない場合に、本パラメータを使用して再生開始同期時刻を設定する

リクエスト例

・画面の右上1/4にPinP再生(Timeline)する場合(音声はMain画面、PinPはループ再生)
pinpPlay PlayerType=Timeline&Source=Playlist01&Scale=50&PositionX=50&PositionY=50&Audio=Main&Loop=ON
・画面全体にPinP(割り込み)再生(NDI)する場合(複数台同期で5秒後再生開始させる場合)
pinpPlay PlayerType=NDI&Source=Device1&Wait=5

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります

失敗時

NG Invalid parameter

※：NGは、以下の要因が考えられます
・パラメータ不正(Invalid parameter)
・Schedule再生以外のプレーヤでの再生時(NDI, Timeline)

	割り込み再生（PinP）停止

コマンド	pinpStop

内容

PinP再生を停止するコマンド
プレーヤ設定がスケジュールの場合のみ動作します

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

pinpStop

リターン

パラメータ名	必須	内容	備考
OK/NG	○	UDPコマンド実行結果	

リターン例

成功時

OK

失敗時

NG

※：NGは、以下の要因が考えられます
・PinP再生状態でない場合

	UDPコマンド実行側とFMP側の時間調整

コマンド	**GetSystemTime**

内容

UDPコマンド実行側PCとFMP側の内部時計の時間差を求めるためのコマンド

なお、時間差はマイクロ秒単位で求め、この値を以降のUDPコマンドのパラメータ「再生開始同期時間」に加算してください

リクエスト

パラメータ名	必須	内容	備考
CurrentTimeVss	○	UDPコマンド実行側時間（Linux時間＋μ秒）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値

リクエスト例

GetSystemTime CurrentTimeVss=1715237197598000

※：UDPコマンドとパラメータの区切り文字は「　」（スペース）で、パラメータの値は「＝」の後になります
※２：1715237197598000＝2024/05/09 06:46:37.598000

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
CurrentTimeVss	○	リクエスト時に設定されたUDPコマンド実行側時間	
CurrentTimeFmp	○	FMP側時間（Linux時間＋μ秒）	Linux時間形式（小数点以下のμ秒を含む）を1000000倍した値

リターン例

成功時

OK CurrentTimeVss=1715237197598000&CurrentTimeFmp=1715237196004200

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「&」、パラメータの値は「＝」の後になります
※２：1715237197598000＝2024/05/09 06:46:37.598000
　　1715237196004200＝2024/05/09 06:46:36.004200

失敗時

なし

	NDIストリーム一覧の取得

コマンド	GetStreamList

内容

NDIストリーム一覧を取得するためのコマンド

リクエスト

パラメータ名	必須	内容	備考
なし			

リクエスト例

```
GetStreamList
```

リターン

パラメータ名	必須	内容	備考
OK	○	UDPコマンド実行結果	本コマンドは「OK」のみ
StreamNum	○	ストリーム数	
Source1		１つ目のソース	※ encodeURIComponent
Status1		１つ目の再生状態(0:Stopped 1:Playing)	
Source2		２つ目のソース	※ encodeURIComponent
Status2		２つ目の再生状態(0:Stopped 1:Playing)	
…			
SourceN		Ｎつ目のソース	※ encodeURIComponent
StatusN		Ｎつ目の再生状態(0:Stopped 1:Playing)	

リターン例

成功時

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

※：実行結果とパラメータの区切り文字は「　」（スペース）で、パラメータ間の区切り文字は「＆」、パラメータの値は「＝」の後になります

失敗時

```
NG Not Support
```

※：NGは、以下の要因が考えられます
　　NDI Sourceが存在しない

●Sample code for PinP Control（python）

PinP制御を行うサンプルコードです。

使用するFMPのIPアドレス（ローカルIP）に併せて「DEFAULT_FMP_IP」「BROADCAST_IP」を変更して使用ください。

```
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 # FMPとの時間差（マイクロ秒）
marge_time_us = 3000000 # FMPの処理時間猶予 (3秒 = 3,000,000マイクロ秒) - 3秒以上推奨

# --- 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):
    """
    GetSystemTimeコマンドを実行し、FMPとの時間差(diff_time_us)を更新します。
    ブロードキャストの場合、最初に応答したFMPの時刻を使用します。
    """
    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):
    """
    GetStreamListコマンドを実行し、利用可能なNDIストリームの一覧を収集します。
    ブロードキャストの場合、すべてのFMPからのリストを統合します。
    """
    command = "GetStreamList"
    responses = execute_command(send_mode, command, target_ip)

    all_ndi_sources = set() # 重複を避けるためにセットを使用

    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)) # ソートしてリストとして返す

# --- 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 ---")

    # プレイヤータイプの選択
    player_type_choice = input("Select PinP Player Type (1: Timeline, 2: NDI): ")
    player_type = ""
    source_name = ""

    if player_type_choice == '1':
        player_type = "Timeline"
        # プレイリスト名のデフォルトを 'Playlist01' に設定
        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) # send_modeとtarget_ipを渡す

        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) # send_modeとtarget_ipを渡す
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 # 現在のターゲットIP
    current_send_mode = '1' # デフォルトはシングル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.")
```