说明

MocapApi是下一代NeuronDataReader（以后简称NDR）的编程接口，目标实现一个跨平台（Win/Mac/Andoid/Ios/Linux等）、跨引擎（u3d/unreal）、用户无代价更新等特性的编程接口，用来接收来自于AxisStudio、Pnlab等软件的socket外发数据。现在已经实现C、C++、C#编程语言的接口，支持Windows平台，u3d和unreal引擎已经支持。

命名规则及调用约定

命名规则

MocapApi有两类特别重要的类型：一类是接口类型，一般是`IMCPXxx`的命名形式；另一类是句柄类型，一般是`MCPXxxHandle_t`的命名形式。其它数据类型基本遵循匈牙利命名法。

`MCPXxxHandle_t`形式的句柄类型，是对象的实体索引，负责组织、管理数据，但并不负责数据的存取。

`IMCPXxx`形式的接口类型，负责根据`MCPXxxHandle_t`形式的句柄类型来存取数据，而不是数据类型本身，并不与数据类型发生关联。

通常`IMCPXxx`形式的接口类型与`MCPXxxHandle_t`形式的句柄类型配合使用。

不同语言的一些差别

- 通常使用C++的接口使用***MocapApi.h***头文件、使用C的接口使用***MocapCApi.h***、C#的接口使用***MocapApi.cs***
- 在以`IMCPXxx`形式命名的接口是在***MocapApi.h***中声明的，通常C++会使用该头文件。与之对应的`MCPXxx_ProcTable`形式命名结构体是C使用的，包含***MocapCApi.h***头文件，表现形式是一个组合了N个函数指针的结构体。在C#中引入***MocapApi.cs***，直接使用`IMCPXxx`形式命名的接口即可。
- C和C++中存在`MCPXxxHandle_t`的句柄类型，C#中没有以`long`替代。

之后关于《类的详细说明》以C++为描述语言。 请参考本节描述转换为相应的编程语言。

接口类型

IMCPXxx形式的接口类型，是不能直接创建的，但可以按需随时获取。C和C++的获取方式接近，C#的获取方式通过静态函数获取。

C和C++的获取方式

IMCPXxx形式的接口类型，是不能直接创建的，要通过`MCPGetGenericInterface`获取，`MCPGetGenericInterface`的声明如下：

```
MCP_INTERFACE EMCPError MCP_CALLTYPE MCPGetGenericInterface(const char *
pchInterfaceVersion,
        void ** ppInterface);
```

- **pchInterfaceVersion**：是一个以空字符结尾的字符串，C++中`IMCPXxx_Version`就可以。

- **ppInterface**： C++中需要传递一个指向`IMCPXxx`指针的指针。

  对应的在C中使用`MCPGetProcessTable`接口获取相应的类型，其实现如下：

```
static EMCPError MCPGetProcessTable(const char * pchInterfaceVersion, void **
ppInterface)
{
    char table[128] = "PROC_TABLE:";
    return MCPGetGenericInterface(strcat(table, pchInterfaceVersion),
ppInterface);
}
```

- **pchInterfaceVersion**： 与C++中相似
- **ppInterface**： 与C++ 中是不同的。在这里则需要传递一个指向`MCPXXX_ProcTable`指针的指针。

C#的获取方式

C#的获取方式要简单的多，直接使用`IMCPXxx.Xxx()`即可。

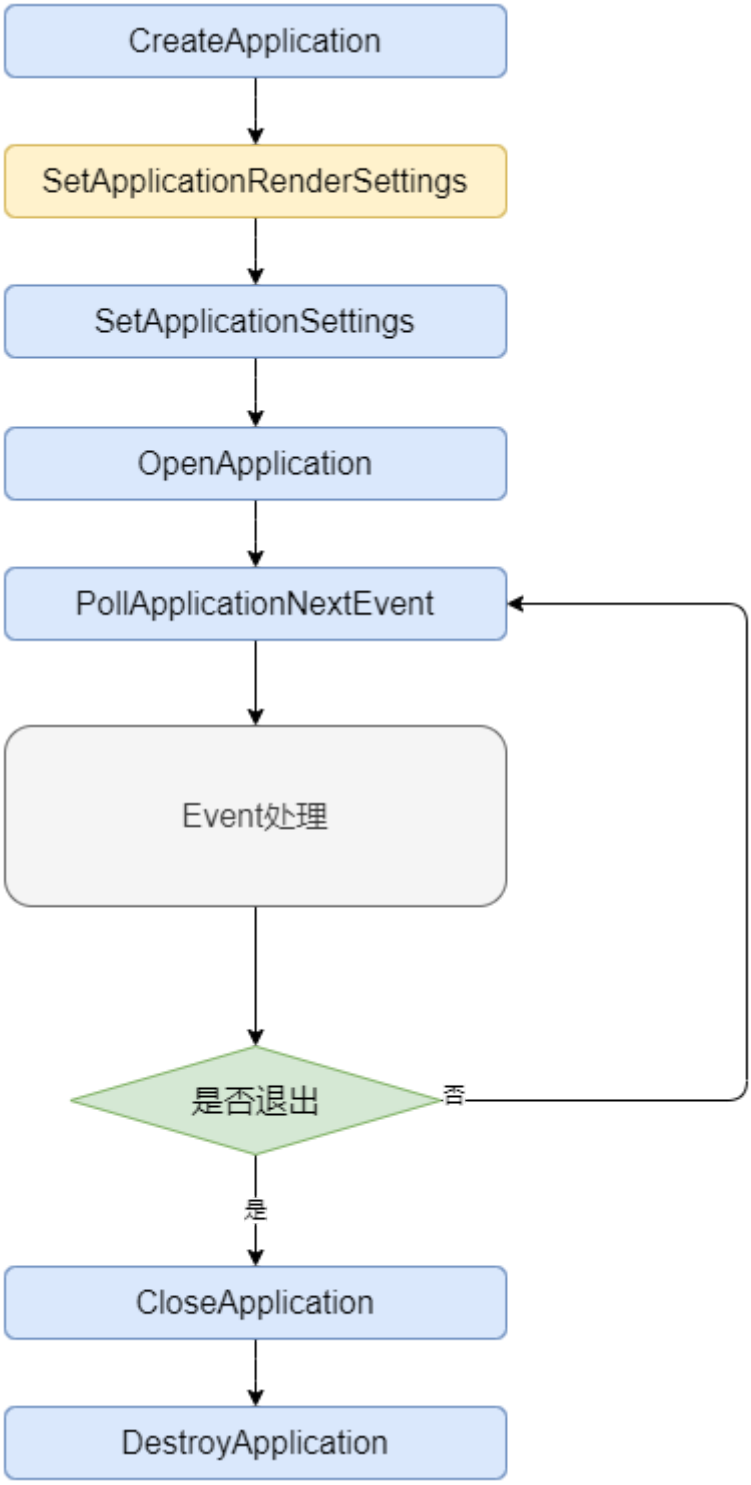
内存管理

MocapApi中遵循谁分配谁释放的原则，即用户一旦调用了`IMCPXxx`的`CreateXxxx`形式的函数，就需要在恰当的时机调用与之对应的`DestroyXxx`函数。`IMCPXxx`的`GetXxx`形式的函数分配的内存由MocapApi自己就释放。

错误处理

MocapApi中有两种错误，一种是函数调用出错，如buffer为空，参数错误等。另一种是程序执行的过程中出错，如与AxisStudio或Pnlab通信出错。MocapApi中的每一个函数都会返回EMCPError的错误码，用户可以根据自己的实际情况去判断。程序执行过程中的错误，会在Event机制中有介绍。

整体调用逻辑



类型的详细说明

IMCPApplication & MCPApplicationHandle_t

一个**Application**对应于Axis Studio/Axis Lab中broadcast的一个输出端口，**Application**封装了用于处理来自于Axis Studio/Axis Lab数据的功能集合。通过调用`PollApplicationNextEvent`轮询最新的Axis Studio 的消息和**Application**自身的状态信息。

使用如下的代码可以获取`IMCPApplication`的指针：

```
MocapApi::IMCPApplication * mcpApplication = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPApplication_Version,
        reinterpret_cast<void **>(&mcpApplication)
```

需要使用`CreateApplication`创建一个**Application**实体，参考代码如下：

```
MocapApi::MCPApplicationHandle_t mcpApplicationHandle = 0;
mcpApplication->CreateApplication(&mcpApplicationHandle);
```

当不再需要某个**Application**时可以使用如下代码：

```
mcpApplication->DestroyApplication(mcpApplicationHandle);
```

CreateApplication

```
virtual EMCPError CreateApplication(MCPApplicationHandle_t *
ulApplicationHandle) = 0;
```

创建一个**Application**实体，当不再需要该实体时需要手动调用`DestroyApplication`，`ulApplicationHandle`返回**Application**实体的句柄。

DestroyApplication

```
virtual EMCPError DestroyApplication(MCPApplicationHandle_t ulApplicationHandle)
= 0;
```

销毁一个**Application**实体，之后`ulApplicationHandle`代表的实体不可再用。

SetApplicationSettings

```
virtual EMCPError SetApplicationSettings(MCPSettingsHandle_t ulSettingsHandle,
MCPApplicationHandle_t ulApplicationHandle) = 0;
```

SetApplicationRenderSettings

```
virtual EMCPError SetApplicationRenderSettings(MCPRenderSettingsHandle_t
ulRenderSettings, MCPApplicationHandle_t ulApplicationHandle) = 0;
```

OpenApplication

```
virtual EMCPError OpenApplication(MCPApplicationHandle_t ulApplicationHandle) =
0;
```

CloseApplication

```
virtual EMCPError CloseApplication(MCPApplicationHandle_t ulApplicationHandle) =
0;
```

GetApplicationRigidBodies

```
virtual EMCPError GetApplicationRigidBodies(
            MCPRigidBodyHandle_t * pRigidBodyHandle,    /*[in, out, optional] */
            uint32_t * punRigidBodyHandleSize,        /*[in, out]*/
            MCPApplicationHandle_t ulApplicationHandle
        ) = 0;
```

GetApplicationAvatars

```
virtual EMCPError GetApplicationAvatars(
            MCPAvatarHandle_t * pAvatarHandle,  /*[in, out, optional] */
            uint32_t * punAvatarHandle,         /*[in, out]*/
            MCPApplicationHandle_t ulApplicationHandle
        ) = 0;
```

参考调用代码如下（其它类似的函数调用机制都可以参考）：

```
numberOfAvatars = 0;
error = application->GetApplicationAvatars(nullptr, &numberOfAvatars,
applicationHandle);
if(error == Error_None){
    avatars = /*分配内存*/
    error = application->GetApplicationAvatars(avatars, &numberOfAvatars,
applicationHandle);
}
```

PollApplicationNextEvent

```
virtual EMCPError PollApplicationNextEvent(
            MCPEvent_t * pEvent /* [in, out,  optional]*/,
            uint32_t * unEvent, /* [in, out] */
            MCPApplicationHandle_t ulApplicationHandle
        ) = 0;
```

其中**pEvent**中每个MCPEvent_t的size都需要赋值

```
MocapApi::MCPEvent_t pEvent[1];
pEvent[0].size = sizeof(MocapApi::MCPEvent_t);
```

如果不使用Cache模式参数调用代码如下：

```
do {
    sizeEvent = 0;
    error = application->PollApplicationNextEvent(nullptr, &sizeEvent,
applicationHandle);
    if(error != Error_None){
        break;  // 处理该错误
    }
    events = /*分配sizeEvent个event*/
    error = application->PollApplicationNextEvent(events, &sizeEvent,
applicationHandle);
} while(error != Error_InsufficientBuffer)
if(error != Error_None){
    // 处理该错误
}
```

GetApplicationSensorModules

```
virtual EMCPError GetApplicationSensorModules(
            MCPSensorModuleHandle_t * pSensorModuleHandle,  /*[in, out,
optional] */
            uint32_t * punSensorModuleHandle,         /*[in, out]*/
            MCPApplicationHandle_t ulApplicationHandle
        ) = 0;
```

GetApplicationTrackers

```
virtual EMCPError GetApplicationTrackers(
            MCPTrackerHandle_t* pTrackerHandle,  /*[in, out, optional] */
            uint32_t* punTrackerHandle,         /*[in, out]*/
            MCPApplicationHandle_t ulApplicationHandle
        ) = 0;
```

QueuedServerCommand

向Server端发送命令，该操作并不会删除该命令。如果在调用该函数之后，调用`DestroyCommand`删除该`cmdHandle`对象，Server端在返回该命令执行结果时不会有Event产生。

```
virtual EMCPError QueuedServerCommand(MCPCommandHandle_t cmdHandle,   /*[in]*/
            MCPApplicationHandle_t ulApplicationHandle) = 0;
```

IMCPAvatar & MCPAvatarHandle_t

Avatar实体对应与AxisStudio 中Avatar，所以只有在与AxisStudio连接时才会有，可以通过`GetAvatarRootJoint`及***Joint***对象的相关函数获取该***Avatar***的层级结构。

使用如下的代码可以获取`IMCPAvatar`的指针：

```
MocapApi::IMCPAvatar * mcpAvatar = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPAvatar_Version,
        reinterpret_cast<void **>(&mcpAvatar)
```

```
typedef uint64_t MCPAvatarHandle_t;
```

GetAvatarIndex

```
virtual EMCPError GetAvatarIndex(uint32_t * index,
            MCPAvatarHandle_t ulAvatarHandle) = 0;
```

GetAvatarRootJoint

Joint是Avatar的骨骼节点，只有在接收BVH数据时才会有。

```
virtual EMCPError GetAvatarRootJoint(MCPJointHandle_t * pJointHandle,
            MCPAvatarHandle_t ulAvatarHandle) = 0;
```

GetAvatarJoints

```
virtual EMCPError GetAvatarJoints(MCPJointHandle_t * pJointHandle, uint32_t *
punJointHandle,
            MCPAvatarHandle_t ulAvatarHandle) = 0;
```

GetAvatarJointByName

```
virtual EMCPError GetAvatarJointByName(const char * name, MCPJointHandle_t *
pJointHandle,
            MCPAvatarHandle_t ulAvatarHandle) = 0;
```

GetAvatarName

```
virtual EMCPError GetAvatarName(const char **,
            MCPAvatarHandle_t ulAvatarHandle) = 0;
```

GetAvatarRigidBodies

RigidBody是与Avatar绑定的刚体，在接收BVH数据时，并且有光混时才会有该数据。

```
virtual EMCPError GetAvatarRigidBodies(MCPRigidBodyHandle_t * pRigidBodies,
uint32_t * punRigidBodies,
            MCPAvatarHandle_t ulAvatarHandle) = 0;
```

IMCPJoint && MCPJointHandle_t

使用如下的代码可以获取 `IMCPJoint` 的指针：

```
MocapApi::IMCPJoint * mcpJoint = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPJoint_Version,
        reinterpret_cast<void **>(&mcpJoint)
```

```
typedef uint64_t MCPJointHandle_t;
```

GetJointName

```
virtual EMCPError GetJointName(const char **,
        MCPJointHandle_t ulJointHandle) = 0;
```

GetJointLocalRotaion

```
virtual EMCPError GetJointLocalRotaion(float * x, float * y, float * z, float *
w,
        MCPJointHandle_t ulJointHandle) = 0;
```

GetJointLocalRotaionByEuler

```
virtual EMCPError GetJointLocalRotaionByEuler(float * x, float * y, float * z,
        MCPJointHandle_t ulJointHandle) = 0;
```

GetJointLocalTransformation

```
virtual EMCPError GetJointLocalTransformation(float * x, float * y, float * z,
        MCPJointHandle_t ulJointHandle) = 0;
```

GetJointDefaultLocalTransformation

```
virtual EMCPError GetJointDefaultLocalTransformation(float * x, float * y, float
* z,
        MCPJointHandle_t ulJointHandle) = 0;
```

GetJointChild

```
virtual EMCPError GetJointChild(MCPJointHandle_t * pJointHandle,
        uint32_t * punJointHandle,
        MCPJointHandle_t ulJointHandle) = 0;
```

GetJointBodyPart

JointBodyPart是Calc数据的描述，只有在接收Calc数据时才会有。

```
virtual EMCPError GetJointBodyPart(MCPBodyPartHandle_t * pBodyPartHandle,
        MCPJointHandle_t ulJointHandle) = 0;
```

GetJointSensorModule

SensorModule是Calc数据的描述，只有在接收Calc数据时才会有。

```
virtual EMCPError GetJointSensorModule(MCPSensorModuleHandle_t*
pSensorModuleHandle,
        MCPJointHandle_t ulJointHandle) = 0;
```

IMCPRigidBody & MCPRigidBodyHandle_t

使用如下的代码可以获取`IMCPRigidBody`的指针：

```
MocapApi::IMCPRigidBody * mcpRigidBody = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPRigidBody_Version,
        reinterpret_cast<void **>(&mcpRigidBody)
```

```
typedef uint64_t MCPRigidBodyHandle_t;
```

GetRigidBodyRotaion

```
virtual EMCPError GetRigidBodyRotaion(float * x, float * y, float * z, float *
w,
        MCPRigidBodyHandle_t ulRigidBodyHandle) = 0;
```

GetRigidBodyTransformation

```
virtual EMCPError GetRigidBodyTransformation(float * x, float * y, float * z,
        MCPRigidBodyHandle_t ulRigidBodyHandle) = 0;
```

GetRigidBodieStatus

```
virtual EMCPError GetRigidBodieStatus(int * status,
        MCPRigidBodyHandle_t ulRigidBodyHandle) = 0;
```

GetRigidBodyId

```
virtual EMCPError GetRigidBodyId(int * id,
        MCPRigidBodyHandle_t ulRigidBodyHandle) = 0;
```

IMCPSensorModule & MCPSensorModuleHandle_t

使用如下的代码可以获取`IMCPSensorModule`的指针：

```
MocapApi::IMCPSensorModule * mcpSensorModule = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPSensorModule_Version,
        reinterpret_cast<void **>(&mcpSensorModule)
```

```
typedef uint64_t MCPSensorModuleHandle_t;
```

GetSensorModulePosture

```
virtual EMCPError GetSensorModulePosture(float * x, float * y, float * z, float
* w,
        MCPSensorModuleHandle_t sensorModuleHandle) = 0;
```

获取姿态四元数。

GetSensorModuleAngularVelocity

```
virtual EMCPError GetSensorModuleAngularVelocity(float * x, float * y, float *
z,
        MCPSensorModuleHandle_t sensorModuleHandle) = 0;
```

GetSensorModuleAcceleratedVelocity

```
virtual EMCPError GetSensorModuleAcceleratedVelocity(float * x, float * y, float
* z,
        MCPSensorModuleHandle_t sensorModuleHandle) = 0;
```

GetSensorModuleId

```
virtual EMCPError GetSensorModuleId(uint32_t * id,
        MCPSensorModuleHandle_t sensorModuleHandle) = 0;
```

GetSensorModuleCompassValue

```
virtual EMCPError GetSensorModuleCompassValue(float * x, float * y, float * z,
    MCPSensorModuleHandle_t sensorModuleHandle) = 0;
```

GetSensorModuleTemperature

```
virtual EMCPError GetSensorModuleTemperature(float * temperature,
        MCPSensorModuleHandle_t sensorModuleHandle) = 0;
```

IMCPBodyPart & MCPBodyPartHandle_t

使用如下的代码可以获取`IMCPBodyPart`的指针：

```
MocapApi::IMCPBodyPart * mcpBodyPart = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPBodyPart_Version,
        reinterpret_cast<void **>(&mcpBodyPart)
```

```
typedef uint64_t MCPBodyPartHandle_t;
```

GetJointPosition

```
virtual EMCPError GetJointPosition(float * x, float * y, float * z,
        MCPBodyPartHandle_t bodyPartHandle) = 0;
```

GetJointDisplacementSpeed

```
virtual EMCPError GetJointDisplacementSpeed(float * x, float * y, float * z,
        MCPBodyPartHandle_t bodyPartHandle) = 0;
```

GetBodyPartPosture

```
virtual EMCPError GetBodyPartPosture(float * x, float * y, float * z, float * w,
        MCPBodyPartHandle_t bodyPartHandle) = 0;
```

IMCPTracker& MCPTrackerHandle_t

Tracker实体对应与Alice Server 中Device，所以只有在与Alice Server连接时才会有。

使用如下的代码可以获取`IMCPTracker`的指针：

```
MocapApi::IMCPTracker * mcpTracker = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPTracker_Version,
        reinterpret_cast<void **>(&mcpTracker)
```

```
typedef uint64_t MCPTrackerHandle_t;
```

GetDeviceCount

```
virtual EMCPError GetDeviceCount(int* devCount,
            MCPTrackerHandle_t ulTrackerHandle) = 0;
```

GetDeviceName

在拿到了DeviceCount的基础上，从0~DeviceCount遍历作为serialNum，获取DeviceName，后续相关数据的获取以DeviceName作为key。

```
virtual EMCPError GetDeviceName(int serialNum, const char** name,
            MCPTrackerHandle_t ulTrackerHandle) = 0;
```

GetTrackerRotation

```
virtual EMCPError GetTrackerRotation(float* x, float* y, float* z, float* w,
const char* deviceName,
            MCPTrackerHandle_t ulTrackerHandle) = 0;
```

GetTrackerPosition

```
virtual EMCPError GetTrackerPosition(float* x, float* y, float* z, const char*
deviceName,
            MCPTrackerHandle_t ulTrackerHandle) = 0;
```

GetTrackerEulerAng

```
virtual EMCPError GetTrackerEulerAng(float* x, float* y, float* z, const char*
deviceName,
            MCPTrackerHandle_t ulTrackerHandle) = 0;
```

SendMessageData

向Alice Server发送信息

```
virtual EMCPError SendMessageData(const char* message, int len,
            MCPTrackerHandle_t ulTrackerHandle) = 0;
```

IMCPRenderSettings & MCPRenderSettingsHandle_t

使用如下的代码可以获取`IMCPRenderSettings`的指针：

```
MocapApi::IMCPRenderSettings * mcpRenderSettings = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPRenderSettings_Version,
        reinterpret_cast<void **>(&mcpRenderSettings)
```

需要使用`CreateRenderSettings`创建一个**RenderSettings**实体，参考代码如下：

```
MocapApi::MCPRenderSettingsHandle_t mcpRenderSettingsHandle = 0;
mcpRenderSettings->CreateRenderSettings(&mcpRenderSettingsHandle);
```

当不再需要某个**RenderSettings**时可以使用如下代码：

```
mcpRenderSettings->DestroyRenderSettings(mcpRenderSettingsHandle);
```

还可以使用`GetPreDefRenderSettings`获取一些预定义的**RenderSettings**实体，但不可以使用`DestroyRenderSettings`销毁。

CreateRenderSettings

```
 virtual EMCPError CreateRenderSettings(MCPRenderSettingsHandle_t *
pRenderSettings) = 0;
```

创建一个**RenderSettings**实体，当不再需要该实体时需要手动调用`DestroyRenderSettings`，`pRenderSettings`返回**RenderSettings**实体的句柄。

GetPreDefRenderSettings

```
virtual EMCPError GetPreDefRenderSettings(EMCPPreDefinedRenderSettings
preDefinedRenderSettings,
        MCPRenderSettingsHandle_t * pRenderSettings) = 0;
```

SetUpVector

```
virtual EMCPError SetUpVector(EMCPUpVector upVector, int sign,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

GetUpVector

```
virtual EMCPError GetUpVector(EMCPUpVector * pUpVector, int * sign,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

SetFrontVector

```
virtual EMCPError SetFrontVector(EMCPFrontVector frontVector, int sign,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

GetFrontVector

```
virtual EMCPError GetFrontVector(EMCPFrontVector * pFrontVector, int * sign,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

SetCoordSystem

```
virtual EMCPError SetCoordSystem(EMCPCoordSystem coordSystem,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

GetCoordSystem

```
virtual EMCPError GetCoordSystem(EMCPCoordSystem * pCoordSystem,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

SetRotatingDirection

```
virtual EMCPError SetRotatingDirection(EMCPRotatingDirection rotatingDirection,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

GetRotatingDirection

```
virtual EMCPError GetRotatingDirection(EMCPRotatingDirection *
pRotatingDirection,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

SetUnit

```
virtual EMCPError SetUnit(EMCPUnit mcpUnit,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

GetUnit

```
virtual EMCPError GetUnit(EMCPUnit * mcpUnit,
        MCPRenderSettingsHandle_t renderSettings) = 0;
```

DestroyRenderSettings

```
virtual EMCPError DestroyRenderSettings(MCPRenderSettingsHandle_t
renderSettings) = 0;
```

销毁一个**RenderSettings**实体，之后`renderSettings`代表的实体不可再用。

IMCPSettings && MCPSettingsHandle_t

使用如下的代码可以获取`IMCPSettings`的指针：

```
MocapApi::IMCPSettings * mcpSettings = nullptr;
MocapApi::MCPGetGenericInterface(MocapApi::IMCPSettings_Version,
        reinterpret_cast<void **>(&mcpSettings)
```

```
typedef uint64_t MCPSettingsHandle_t;
```

需要使用`CreateApplication`创建一个**Settings**实体，参考代码如下：

```
MocapApi::MCPSettingsHandle_t mcpSettingsHandle = 0;
mcpSettings->CreateSettings(&mcpSettingsHandle);
```

当不再需要某个**Settings**时可以使用如下代码：

```
mcpSettings->DestroySettings(mcpSettingsHandle);
```

CreateSettings

```
virtual EMCPError CreateSettings(MCPSettingsHandle_t * pSettingsHandle) = 0;
```

创建一个**Settings**实体，当不再需要该实体时需要手动调用`DestroySettings`，`pSettingsHandle`返回**Settings**实体的句柄。

DestroySettings

```
virtual EMCPError DestroySettings(MCPSettingsHandle_t ulSettingsHandle) = 0;
```

销毁一个**Settings**实体，之后`ulSettingsHandle`代表的实体不可再用。

SetSettingsUDP

```
virtual EMCPError SetSettingsUDP(uint16_t localPort,
        MCPSettingsHandle_t ulSettingsHandle) = 0;
```

SetSettingsUDPServer

以UDP建立连接时，需要设置Server相关的IP和端口。

```
virtual EMCPError SetSettingsUDPServer(const char* serverIp, uint16_t
serverPort,
            MCPSettingsHandle_t ulSettingsHandle) = 0;
```

SetSettingsTCP

```
virtual EMCPError SetSettingsTCP(const char * serverIp, uint16_t serverPort,
        MCPSettingsHandle_t ulSettingsHandle) = 0;
```

SetSettingsBvhRotation

```
virtual EMCPError SetSettingsBvhRotation(EMCPBvhRotation bvhRotation,
        MCPSettingsHandle_t ulSettingsHandle) = 0;
```

SetSettingsBvhTransformation

```
virtual EMCPError SetSettingsBvhTransformation(EMCPBvhTransformation
bvhTransformation,
        MCPSettingsHandle_t ulSettingsHandle) = 0;
```

SetSettingsBvhData

```
virtual EMCPError SetSettingsBvhData(EMCPBvhData bvhData,
            MCPSettingsHandle_t ulSettingsHandle) = 0;
```

SetSettingsCalcData

```
virtual EMCPError SetSettingsCalcData(MCPSettingsHandle_t ulSettingsHandle) = 0;
```

IMCPCommand

MocapApi向Server端发送的命令接口，对应的命令句柄为`MCPCommandHandle_t`。

CreateCommand

根据命令标记创建命令对象。

```
 virtual EMCPError CreateCommand(uint32_t cmd, MCPCommandHandle_t* handle_) = 0;
```

SetCommandExtraFlags

设置额外的命令标记。

```
virtual EMCPError SetCommandExtraFlags(uint32_t extraFlags, MCPCommandHandle_t
handle_) = 0;
```

SetCommandExtraLong

设置命令的参数。

```
virtual EMCPError SetCommandExtraLong(uint32_t extraLongIndex, intptr_t
extraLong,
            MCPCommandHandle_t handle_) = 0;
```

GetCommandResultMessage

获取Server端返回的错误消息。

```
virtual EMCPError GetCommandResultMessage(const char ** pMsg, MCPCommandHandle_t
handle_) = 0;
```

GetCommandResultCode

获取Server端返回的错误码。

```
virtual EMCPError GetCommandResultCode(uint32_t *pResCode, MCPCommandHandle_t
handle_) = 0;
```

GetCommandProgress

获取Server端返回的命令执行进度信息。

```
virtual EMCPError GetCommandProgress(uint32_t progress, intptr_t extra,
MCPCommandHandle_t handle_) =0;
```

DestroyCommand

```
virtual EMCPError DestroyCommand(MCPCommandHandle_t handle_) = 0;
```

IMCPCalibrateMotionProgress

人体姿态校准进度接口，对应的MCPCalibrateMotionProgressHandle_t对象。通过`IMCPCommand::GetCommandProgress(CommandProgress_CalibrateMotion, ...)`获取。

GetCalibrateMotionProgressCountOfSupportPoses

本次校准所需要做的校准姿势数。

```
virtual EMCPError GetCalibrateMotionProgressCountOfSupportPoses(uint32_t *
pCount,
            MCPCalibrateMotionProgressHandle_t handle_) = 0;
```

GetCalibrateMotionProgressNameOfSupportPose

获取本次校准的校准姿势名。

```
virtual EMCPError GetCalibrateMotionProgressNameOfSupportPose(char* name,
uint32_t* pLenOfName,
        uint32_t index, MCPCalibrateMotionProgressHandle_t handle_) = 0;
```

GetCalibrateMotionProgressStepOfPose

获取指定校准姿势的校准阶段。

```
virtual EMCPError GetCalibrateMotionProgressStepOfPose(uint32_t* pStep,
        const char*name, MCPCalibrateMotionProgressHandle_t handle_) = 0;
```

GetCalibrateMotionProgressCountdownOfPose

获取指定校准姿势的倒计时。

```
virtual EMCPError GetCalibrateMotionProgressCountdownOfPose(uint32_t*
pCountdown,
        const char*name, MCPCalibrateMotionProgressHandle_t handle_) = 0;
```

GetCalibrateMotionProgressProgressOfPose

获取指定校准姿势的校准进度。

```
virtual EMCPError GetCalibrateMotionProgressProgressOfPose(uint32_t* pProgress,
        const char*name, MCPCalibrateMotionProgressHandle_t handle_) = 0;
```

GetCalibrateMotionProgressStepOfCurrentPose

获取当前校准姿势和校准阶段。

```
virtual EMCPError GetCalibrateMotionProgressStepOfCurrentPose(uint32_t * pStep,
        char* name, uint32_t * pLenOfName, MCPCalibrateMotionProgressHandle_t
handle_) = 0;
```

GetCalibrateMotionProgressCountdownOfCurrentPose

获取当前校准姿势和校准倒计时。

```
virtual EMCPError GetCalibrateMotionProgressCountdownOfCurrentPose(uint32_t*
pCountdown,
        char* name, uint32_t* pLenOfName, MCPCalibrateMotionProgressHandle_t
handle_) = 0;
```

GetCalibrateMotionProgressProgressOfCurrentPose

获取当前校准执行和校准进度。

```
virtual EMCPError GetCalibrateMotionProgressProgressOfCurrentPose(uint32_t*
pProgress,
        char* name, uint32_t* pLenOfName, MCPCalibrateMotionProgressHandle_t
handle_) = 0;
```

EMCPEventType

```
enum EMCPEventType
{
        MCPEvent_None,
        MCPEvent_AvatarUpdated,
        MCPEvent_RigidBodyUpdated,
        MCPEvent_Error,
        MCPEvent_SensorModulesUpdated,
        MCPEvent_TrackerUpdated,
        MCPEvent_CommandReply,
};
```

EMCPBvhRotation

```
enum EMCPBvhRotation
{
        BvhRotation_XYZ,
        BvhRotation_XZY,
        BvhRotation_YXZ,
        BvhRotation_YZX,
        BvhRotation_ZXY,
        BvhRotation_ZYX,
};
```

EMCPBvhData

```
enum EMCPBvhData
{
        BvhDataType_String,
        BvhDataType_BinaryWithOldFrameHeader,
        BvhDataType_Binary,
        BvhDataType_Mask_LegacyHumanHierarchy,
};
```

EMCPBvhTransformation

```
enum EMCPBvhTransformation
{
        BvhTransformation_Disable,
        BvhTransformation_Enable,
};
```

EMCPUpVector

```
enum EMCPUpVector
{
        UpVector_XAxis = 1,
        UpVector_YAxis = 2,
        UpVector_ZAxis = 3
};
```

EMCPFrontVector

```
enum EMCPFrontVector
{
        FrontVector_ParityEven = 1,
        FrontVector_ParityOdd = 2
};
```

EMCPCoordSystem

```
enum EMCPCoordSystem
{
        CoordSystem_RightHanded,
        CoordSystem_LeftHanded
};
```

EMCPRotatingDirection

```
enum EMCPRotatingDirection
{
        RotatingDirection_Clockwise,
        RotatingDirection_CounterClockwise,
};
```

EMCPPreDefinedRenderSettings

```
enum EMCPPreDefinedRenderSettings
{
        PreDefinedRenderSettings_Default,
        PreDefinedRenderSettings_UnrealEngine,
        PreDefinedRenderSettings_Unity3D,
        PreDefinedRenderSettings_Count,
};
```

EMCPUnit

```
enum EMCPUnit
{
        Unit_Centimeter,
        Uint_Meter,
};
```

EMCPReplay

```
enum EMCPReplay
{
        MCPReplay_Response,
        MCPReplay_Running,
        MCPReplay_Result,
};
```

EMCPCommand

```
enum EMCPCommand
{
        CommandStartCapture,
        CommandStopCapture,
        CommandZeroPosition,
        CommandCalibrateMotion,
        CommandStartRecored,
        CommandStopRecored,
        CommandResumeOriginalPosture,
};
```

EMCPCommandStopCatpureExtraFlag

```
enum EMCPCommandStopCatpureExtraFlag
{
        StopCatpureExtraFlag_SensorsModulesPowerOff,
        StopCatpureExtraFlag_SensorsModulesHibernate,
};
```

EMCPCommandExtraLong

```
enum EMCPCommandExtraLong
{
        CommandExtraLong_DeviceRadio,
        CommandExtraLong_AvatarName,
};
```

EMCPCommandProgress

```
enum EMCPCommandProgress
{
        CommandProgress_CalibrateMotion,
};
```

EMCP_CalibrateMotionProgressStep

```
 enum EMCPCalibrateMotionProgressStep
{
        CalibrateMotionProgressStep_Prepare,
        CalibrateMotionProgressStep_Countdown,
        CalibrateMotionProgressStep_Progress,
};
```

MCPEvent_t

```
struct MCPEvent_t
{
    uint32_t        size;
    EMCPEventType   eventType;
    double          fTimestamp;
    MCPEventData_t  eventData;
};
```

成员说明:

- **size** : 输入值，必须是sizeof(MCPEvent_t)
- **eventType** : 输出值
- **fTimestamp** : 自动软件启动以来的时间值
- **eventData** : 事件的详细信息

MCPEventData_t

```
union MCPEventData_t
{
    MCPEvent_Reserved_t reserved;

    MCPEvent_MotionData_t motionData;

    MCPEvent_SystemError_t systemError;

    MCPEvent_SensorModuleData_t sensorModuleData;

    MCPEvent_TrackerData_t trackerData;

    MCPEvent_CommandRespond_t commandRespond;
};
```

MCPEvent_Reserved_t

```
struct MCPEvent_Reserved_t
{
    uint64_t reserved0;
    uint64_t reserved1;
    uint64_t reserved2;
    uint64_t reserved3;
    uint64_t reserved4;
    uint64_t reserved5;
};
```

MCPEvent_MotionData_t

```
struct MCPEvent_MotionData_t
{
    MCPAvatarHandle_t   avatarHandle;
};
```

MCPEvent_SystemError_t

```
struct MCPEvent_SystemError_t
{
    EMCPError error;
};
```

MCPEvent_SensorModuleData_t

```
struct MCPEvent_SensorModuleData_t
{
    MCPSensorModuleHandle_t _sensorModuleHandle;
};
```

MCPEvent_TrackerData_t

```
struct MCPEvent_TrackerData_t
{
    MCPTrackerHandle_t _trackerHandle;
};
```

MCPEvent_CommandRespond_t

```
struct MCPEvent_CommandRespond_t
{
        MCPCommandHandle_t _commandHandle;
        EMCPReplay _replay;
};
```