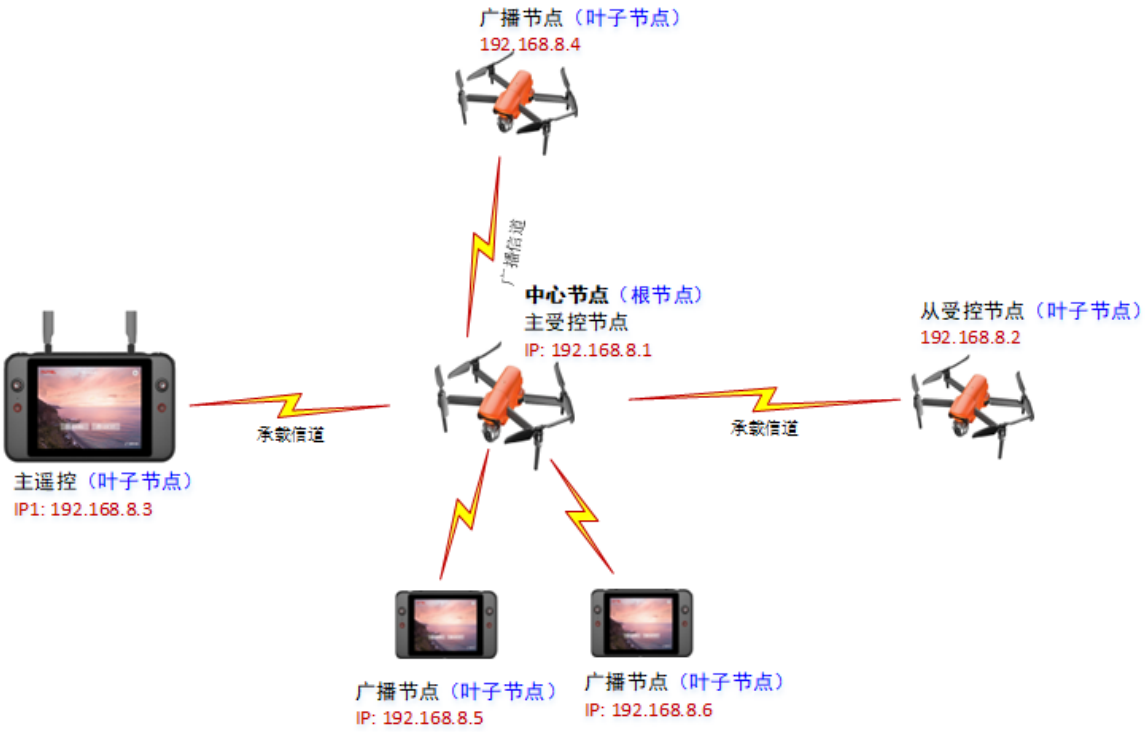
# Autel Robotics 道通智能组网MSDK V2.5说明文档

## 0、概述

MSDK新版增加了组网功能，支持将多遥控器（主、从遥控器）、多飞机加入到同一个网络中。组网成功后，所有的设备中有且只有一个主遥控器和一个中继飞机，其他遥控器都是从遥控器，其它飞机为普通节点飞机。

群组是建立在组网基础上的业务，加入到同一个群分飞机称为群组，可以进行群控，未加入群组的飞机作为单机，可以选择控制模式（单控、全控、群控）对飞机进行各种操作。

### 0.1 组网模型




星型网络

- **名词解释**

| 概念 | 说明 | |
| --- | --- | --- |
| 组网 | 设备加入到一个网络 | |
| 群组 | 已经组网的无人机加入群组 | |
| DeviceId | 无人机唯一ID | |
| NodeId | 本次组网的节点ID | |
| 中继机 | 中心节点飞机 | |
| 主遥控器 | 发起组网的遥控器 | |
| 从遥控器 | 加入组网的遥控器 | |
| 单控 | 选择一架无人机并控制 | |
| 群控 | 选择一个群组进行控制 | |
| 全控 | 控制所有无人机 | |
| watch设备 | 能够查看图传码流的设备 | |

- **组网角色与权限**

| 角色 | 控制权限 | 角色切换 | 多码流 |
| --- | --- | --- | --- |
| 主遥控器 | 全部权限 | 不能切换 | 可以观看受控飞机的码流 |
| 从遥控器 | 摇杆不可用 | 不能切换 | 可以观看中受控飞机的码流 |
| 中继飞机 | - | 可以通过中继替换流程替换 | 可以向所有遥控器推送码流 |
| 叶子飞机 | - | 可以切换为受控飞机 | 不可推送码流 |
| 受控飞机 | - | 可以切换为叶子飞机 | 可以向所有遥控器推送码流 |

# 1、组网相关接口

## 1.1 DeviceManager

DeviceManager负责各种设备的创建管理，通过DeviceManager.getMultiDeviceOperator()获组网相关操作接口

```
interface IMultiDeviceOperator
    /**
     * 获取群组操作接口
     */
    fun getGroupMeshApi():IGroupMeshApi
    /**
     * 获取组网操作接口
     */
    fun getNetMeshManager(): INetMeshManager
    /**
     * 是否单控
     */
```

```
    fun isSingleControl(): Boolean

    /**
     * 是否是组网版本
     */
    fun isNetMesh(): Boolean

    /**
     * 组网：飞机控制变化监听
     */
    fun addControlChangeListener(listener: IControlDroneListener)
    fun removeControlChangeListener(listener: IControlDroneListener)
    /**
     * 组网：watch飞机变化监听
     */
    fun addWatchChangeListener(listener: IWatchDroneListener)
    fun removeWatchChangeListener(listener: IWatchDroneListener)
    /**
     * 获取飞行器列表
     * @return autel drone device list
     */
    fun getDroneDevices(): List<IAutelDroneDevice>
    /**
     * 组网控制模式
     */
    fun getControlMode(): ControlMode
    /**
     * 存在群组信息
     */
    fun hasGroupMesh(): Boolean
    /**
     * 获取遥控器列表
     * @return autel remote device list
     */
    fun getRemoteDevices(): List<IAutelRemoteDevice>

    /**
     * 通过设备id获取飞行器
     * @param deviceId device id
     * @return drone device object with id
     */
    fun getDroneDeviceById(deviceId: Int): IAutelDroneDevice?
    /**
     * 通过设备nodeId获取飞行器
     * @param nodeId  id
     * @return drone device object with id
     */
    fun getDroneDeviceByNodeId(nodeId: Int): IAutelDroneDevice?
    /**
     * 本地遥控器信息
     * @param deviceId device id
     * @return remote device object with id
     */
    fun getLocalRemoteDevice(): RemoteDevice
    /**
     * 获取已经连接飞机的id集合
     * @return connected devices id list
     */
```

```
    fun getConnectedDeviceIds(): List<Int>
    /**
     * 获取飞机升级设备列表
     * @return base devices list
     */
    fun getDroneUpgradeDevices(): MutableList<IBaseDevice>
    /**
     * 根据deviceIdList得到飞机设备的keyManagerList
     */
    fun generateKeyManagerList(deviceIdList: List<Int>):
MutableList<IKeyManager>
    /**
     * 执行多机命令接口
     */
    fun <Param, Result> performActionList(
        deviceIdList: List<Int>, key: AutelKey.ActionKey<Param, Result>, param:
Param,
        callback:
DeviceManager.CompletionCallbackWithParam<DeviceManager.DeviceActionResult<Resul
t>>?
    )
    /**
     * 组网：多机监听接口
     */
    fun <Result> addDroneDevicesListener(key: AutelKey<Result>, callbacks:
DeviceManager.KeyManagerListenerCallBack)
    /**
     * 组网： 多机取消监听接口
     */
    fun <Result> removeDroneDevicesListener(key: AutelKey<Result>, callbacks:
DeviceManager.KeyManagerListenerCallBack)
    /**
     * 组网：获取中继飞机
     */
    fun getCenterDroneDevice(): IAutelDroneDevice?
    /**
     * 组网：获取所控的在线飞机列表
     */
    fun getControlledDroneList(): MutableList<IAutelDroneDevice>
    /**
     * 组网：获取watch的在线飞机列表
     */
    fun getWatchedDroneList(): MutableList<IAutelDroneDevice>

    /**
     * 组网：是否是主遥控器
     */
    fun isMainRC():Boolean
    /**
     * 组网：设备及控制变化监听
     */
    fun addNetMeshChangeListener(listener: IMeshDeviceChangedListener)
    /**
     * 组网：设备及控制变化监听
     */
    fun removeNetMeshChangeListener(listener: IMeshDeviceChangedListener)
    /**
     * 组网：群组列表
```

```
     */
    fun getGroupList():List<GroupDeviceData>
    /**
     * 组网：单机列表（未加入群组）
     */
    fun getSingleDeviceList(): List<IAutelDroneDevice>
    /**
     * 获取受控飞机相册端口
     */
    fun getActiveDroneAlbumPort():Int
    /**
     * 获取受控飞机相册端口
     */
    fun getActiveFileServicePort():Int
    /**
     * 获取受控飞机的相册服务地址前缀
     */
    fun getActiveAlbumBaseUrl():String

    /**
     * 获取受控飞机的文件服务地址前缀
     */
    fun getActiveFileBaseUrl():String
}
```

## 1.2 组网相关接口

主要提供组网相关接口：启动组网、添加、删除、重命名设备，设置中继设备，设置Watch设备，解散组网等功能

```
interface INetMeshManager {

    /**
     * 获取设备信息
     */
    fun getAllMeshDeviceList():List<DeviceInfoBean>

    /**
     * 本地遥控器名称
     */
    fun getLocalRCName():String?

    /**
     * 主遥控器名称
     */
    fun getMainRCName():String?

    /**
     * 主遥控器设置的watch设备
     */
    fun getMainRcWatchDrone():IAutelDroneDevice?

    /**
     * 是否正在组网中
     */
    fun isNetMeshing():Boolean
```

```
    /**
     * 是否解散了组网
     */
    fun isMeshDisband():Boolean

    /**
     * 开始组网
     * @param bean
     * @param startMatch false：超时后继续 ，true：正常启动
     * @param callback
     */
    fun startNetMeshMatching(
        bean: CreateDeviceNetworkReq,
        callback:
CommonCallbacks.CompletionCallbackWithParam<CreateDeviceNetworkResp>
    )

    /**
     * 完成组网
     */
    fun completeNetMeshMatching(
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 剔除组网设备
     * */
    fun delNetMeshDevice(
        deviceId: Int,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 设置为中心结点
     */
    fun setCenterNode(
        deviceId: Int,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 加入组网
     */
    fun joinDeviceNetMesh(
        joinReq: JoinDeviceNetworkReq?,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 解散组网团队
     */
    fun disbandNetMesh(
        groupId: Long?,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     *退出组网团队
```

```
     */
    fun quitNetMeshMatching(
        groupId: Long?,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 设置本地设备在组网团队的名称
      */
    fun nameDeviceNetMeshMatching(
        editDeviceNameReq: EditDeviceNameReq,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 多控模式切换图传，参数 DeviceId
     * return : 1：成功 2：失败 0：未知
     * */
    fun setWatchDevice(
        selectDevice: List<Int>?,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
    * 组网流控设置
    */
    fun setNetMeshStreamControl(streamList: List<CameraStreamInfo>, callback:
CommonCallbacks.CompletionCallbackWithParam<Int>)
}
```

## 1.3 群组相关接口

主要用于创建或解散群组，设置控制模式（单机控制、群组控制、全控）

```
interface IGroupMeshApi {

    /***
     * 为指定飞机列表创建一个群组
     */
    fun createGroup(
        nodeIds: List<Int>,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 添加飞机到群组
     * @param groupId 群组ID
     * @param nodeIds 飞机nodeId数组
     */
    fun addDroneToGroup(
        groupId: Int,
        nodeIds: List<Int>,
        callback: CommonCallbacks.CompletionCallbackWithParam<Int>
    )

    /**
     * 从群组删除飞机
```

```
     * @param groupId 群组ID
     * @param nodeIds 飞机nodeId数组
     */
    fun delDroneFromGroup(
        groupId: Int,
        nodeIds: List<Int>,
        callback: CommonCallbacks.CompletionCallbackWithParam<Void>
    )

    /**
     * 解散群组
     * @param groupId 群组ID
     */
    fun disbandGroup(
        groupId: Int,
        callback: CommonCallbacks.CompletionCallbackWithParam<Void>?
    )

    /**
     * 选择群组或者单机
     * @param mode [ControlMode]
     * @param id  群组控制：groupId;
     *            单机控制：NodeId;
     *            全选：此参数被忽略
     */
    fun switchControlMode(
        mode: ControlMode,
        id: Int,
        callback: CommonCallbacks.CompletionCallbackWithParam<Void>
    )

    /**
     * 修改群组名称
     * @param groupId 群组ID
     * @param name 群组名称
     */
    fun changeGroupName(
        groupId: Int,
        name: String,
        callback: CommonCallbacks.CompletionCallbackWithParam<Void>
    )

    /**
     * 设置长机
     * @param groupId 群组ID
     * @param deviceNodeId 设备节点ID
     */
    fun setGroupDroneLeader(
        groupId: Int,
        deviceNodeId: Int,
        callback: CommonCallbacks.CompletionCallbackWithParam<Void>
    )
```

# 2、设备监听

## 2.1 飞机事件监听

提供了针对多机组网的设备事件监听，方便多机情况下事件的监听

```
DeviceManager.getDeviceManager().addDroneDevicesListener(reportKey,
devicesListener)
DeviceManager.getDeviceManager().removeDroneDevicesListener(reportKey,
devicesListener)

private val devicesListener = object : DeviceManager.KeyManagerListenerCallBack
{
    override fun onListenerValueChanged(value:
DeviceManager.DeviceListenerResult<*>) {
        (value.result as? MissionWaypointStatusReportNtfyBean)?.let { bean ->
            missionStatusMap[value.drone.getDeviceNumber()] = bean
        }
    }
}
```

# 3、飞机能力提供

## 3.1 飞机能力均可通过飞机设备获取

```
interface IAutelDroneDevice : IBaseDevice {

    /**
     * 获取相册Manager
     * @return 获取相册Manager
     */
    fun getAlbumManager(): IAlbumManager

    /**
     * 获取航点任务Manager
     * @return 获取航点任务Manager
     */
    fun getWayPointMissionManager(): IMissionManager

    /**
     * 获取任务追踪管理类
     * @return 获取任务追踪管理类
     */
    fun getTrackMissionManager(): ITrackMissionManager

    /**
     * 获取相机能力集管理类
     * @return 获取相机能力集管理类
     */
    fun getCameraAbilitySetManger(): ICameraAbilitySetManager

    /**
     * 获取当前设备的云台类型
     * @return 获取当前设备的云台类型
     */
    fun getGimbalDeviceType(): GimbalTypeEnum

    /**
     * 获取播放器管理类
```

```
     */
    fun getAutelPlayerManager(): AutelPlayerManager

    /**
     * 获取文件服务管理类
     */
    fun getFileServiceManager(): FileServiceManager

    /**
     * 获取设备缓存数据
     */
    fun getDeviceStateData(): DroneStateData


    /**
     * 获取RTK管理类
     * @return 获取RTK管理类
     */
    fun getRtkManager(): IRTKManager

    /**
     * 获取飞机连接状态
     * @return 获取飞机连接状态
     */
    fun isConnected(): Boolean {
        return true
    }

    /**
     * 更新飞机连接状态
     * @param connect setting connection status for drone device with RCPad
     */
    fun setConnectState(connect: Boolean) {}

    /**
     * 飞机起飞状态
     */
    fun isPreFlightOK(): Boolean

    /**
     * 当前飞机的类型
     */
    fun getDroneType(): DroneType

    /**
     * 飞机是否受控
     */
    fun isControlled(): Boolean

    fun getNodeId(): Int?

    fun getGroupId():Int

    /**
     * 设备是否 watch 状态
     */
    fun isWatched(): Boolean
```

```
    /**
     * 是否为长机
     */
    fun isLeader():Boolean

    /**
     * 是否为中继节点
     */
    fun isCenter():Boolean

    /**
     * 未加入任何群组
     */
    fun isSingle():Boolean

    /**
     * 当前飞机IP
     */
    fun getIp(): String { return "" }

    /**
     * 相册服务端口
     */
    fun getAlbumPort(): Int { return 0}

    /**
     * 文件服务端口
     */
    fun getFileServicePort(): Int {return 0}

    fun getAlbumBaseUrl():String{ return "" }

    fun getProgressQueryUrl():String{ return "" }

    fun getMissionUploadUl():String{ return ""}

    fun getFileBaseUrl(): String {return ""}
}
```

# 4、Autel组网功能示例

## 4.1 组网主要接口

### 4.1.1 主遥控器

- 开始组网配对：30s超时，超时后可继续向启动组网

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().startNetMesh
Matching(bean,callback)
```

- 剔除设备：中继设备，无法删除，删除中继设备等于解散组网，调用前需要判断是否要删除的是中继设备，如果是中继设备，则直接调用解散组网

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().delNetMeshDe
vice(id,callback)
```

- 设置中继

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().setCenterNod
e(id,callback)
```

- 完成组网：完成组网后会重置网络，所有设备连接会短暂断开然后重新连接

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().completeNetM
eshMatching(callback)
```

- 解散组网：会删除所有设备

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().disbandNetMe
sh(groupdId,callback)
```

- 修改组网设备名称

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().nameDeviceNe
tMeshMatching(req,callback)
```

### 4.1.2 从遥控器

- 加入组网

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().joinDeviceNe
tMesh(req,callback)
```

- 退出组网

```
DeviceManager.getMultiDeviceOperator().getNetMeshManager().quitNetMeshM
atching（groupdId,callback）
```

## 4.2 组网监听

- 设备状态监听(设备生命周期变化监听，基线sdk中，sdk初始化时，默认创建了一个设备，组网sdk中默认没有飞机设备，所有的飞机设备，都依赖SDK创建，并会回调出来。如果没有创建设备，则DeviceManager下的设备列表为空)[主要用在上层设备对象回收、创建等]

  依次是：设备创建->设备连接变化->设备主服务可用变化->设备能力集变化->设备销毁

```
interface IAutelDroneListener {
/**
     * 组网设备创建
     * @param drone drone device
     */
    fun onDroneCreate(drone: IAutelDroneDevice){}

     /**
     * 无人机设备连接状态的监听器
     *
     * @param connected 已连接 - is connected
     * @param drone 无人机装置 - drone device
```

```
     */
    fun onDroneChangedListener(connected: Boolean, drone:
IAutelDroneDevice){}

    /**
     * 主服务是否可用
     * @param valid true 可用，false 不可用
     * @param drone drone device
     */
    fun onMainServiceValid(valid: Boolean, drone: IAutelDroneDevice) {}

    /**
     *相机能力集变更通知
     * @param localFetched true 本地能力集解析成功
     * @param remoteFetched true 远程能力集解析成功
     */
    fun onCameraAbilityFetchListener(localFetched: Boolean,
remoteFetched: Boolean, drone: IAutelDroneDevice){}

    /**
     * SDK业务接口报错回调
     * @param errorInfo
     */
    fun onSDKErrorListener(errorInfo: SDKErrorUtil) {}

    /**
     * 组网设备销毁
     * @param drone drone device
     */
    fun onDroneDestroy(drone: IAutelDroneDevice){}

}
```

- 设备控制变化(组网过程中触发，用户下发控制切换后触发)

```
DeviceManager.getMultiDeviceOperator()
.addControlChangeListener(object : IControlDroneListener {
    override fun onControlChange(mode: ControlMode, droneList:
List<IAutelDroneDevice>){
    }
})
```

- 设备watch变化[有的图传版本支持两个设备watch，中继飞机和另一个飞机，有的图传版本只支持单设备watch.]

组网中最多支持两个飞机的视频同时播放，AutelPlayerManager中一共有中继飞机和从飞机的6个镜头的播放器。获取到播放器后，将播放器对应的渲染控件移除，再将你希望的渲染控件加入进去，则可以渲染到对应的控件上

```
DeviceManager.getMultiDeviceOperator().addWatchChangeListener(object:object :
IWatchDroneListener {
    override fun onWatchChange(droneList: List<IAutelDroneDevice>) {
        if(droneList.contans(currentDrone)){//如果当前设备被watch了
            //播放器显示当前飞机的流，通过飞机拿到播放器端口号，通过端口号拿到对应播放器
        val playerId = DeviceUtils.getPlayerId(droneDevice, lensTypeEnum)
        if (playerId != null) {
```

```
            val autelPlayer =
AutelPlayerManager.getInstance().getAutelPlayer(playerId)

            if (autelPlayer != null) {
                autelPlayer?.removeVideoView()
            }
            autelPlayer?.addVideoView(renderView)

            AutelLog.i("SwitchDrone", "startPlay = " + autelPlayer + " port:" +
playerId + " view :" + renderView.toString())
            getCurrentAutelPlayerKeyFrame(playerId)
        }
    }
})
```