Xiaomi Wear 타사 앱 기능 오픈 인터페이스 Document_v1.4

버전 1.4 업데이트 로그

안드로이드 R과 호환 가능

버전 1.3 업데이트 로그

향후 Xiaomi Wear 및 Xiaomi Health 합병 프로젝트와 호환되며 API는 변경되지 않으며 SDK를 직접 교체하면 됩니다.

버전 1.2 업데이트 로그:

jdk1.7 버전과 호환되며 API는 변경되지 않았습니다.

버전 1.1 업데이트 로그:

1. 상태 구독 및 상태 쿼리 관련 인터페이스의 반환 값을 수정합니다. 자세한 내용은 문서의 섹션 3.2 및 3.3을 참조하세요.

2. 메시지 알림 API를 추가합니다. 자세한 내용은 문서의 5부를 참조하세요.

1. 연결된 웨어러블 기기 쿼리(권한 필요 없음)

```
//NodeApi 객체를 먼저 가져옵니다.
NodeApi api = Wearable.getNodeApi(context); // 연결된 장치를 얻기 위
해 getConnectedNodes 메서드를 호출합니다 .
api.getConnectedNodes().addOnSuccessListener(new OnSuccessListener<List<Node>>() {

        @Override
        public void onSuccess(List< 노드> 노드) {
            //현재 연결된 장치를 가져옵니다. 현재는 한 번에 하나의 장치만 연결할 수 있습니다.
        }
}).addOnFailureListener(새로운 OnFailureListener() {
        @보수
        public void onFailure(@NonNull 예외 e) {
            //연결된 장치를 가져오지 못했습니다.
        }
});
```

2. 권한 조회 및 신청

2.1 인증상태 조회

승인을 위한 첫 번째 호출은 기본적으로 Permission.DEVICE_MANAGER 및 Permission.Notify 권한을 부여합니다.

```
//AuthApi 객체를 먼저 가져옵니다.
AuthApi authApi = Wearable.getAuthApi(context); //checkPermission 메소
드를 호출하여 인증 상태를 얻습니다
authApi.checkPermission("nodeId",Permission.DEVICE_MANAGER)
                .addOnSuccessListener(새로운 OnSuccessListener<Boolean>() {
                    @Override
                    public void onSuccess(Boolean result) {
                        //승인된 경우 결과는 true이고, 승인되지 않은 경우 결과는 false입니다.
                    }
                }).addOnFailureListener(새로운 OnFailureListener() { @Override
```

```
                public void onFailure(@NonNull 예외 e ) {
                    //인증 획득 실패
                }
}); //
권한 집합 정의
권한[] 권한 = 새 권한[]
{Permission.DEVICE_MANAGER,Permission.NOTIFY}; //
checkPermissions 인터페이스를 호출하여 권한 인증 상태 세트를 쿼리
합니다. authApi.checkPermissions("nodeId",permissions)
        .addOnSuccessListener(새로운 OnSuccessListener<Boolean[]>() {
            @Override
            public void onSuccess(Boolean[] results) {
                //승인된 경우 결과는 true, 승인되지 않은 경우 결과는 false이며 요청한 순서대로 결과가 반환됩니다.
            }
        }).addOnFailureListener(새로운 OnFailureListener() {
            @보수
            public void onFailure(@NonNull Exception e) { //인증 상태 획득 실패

            }
});
```

2.2 허가 신청

```
//AuthApi 객체를 먼저 가져옵니다.
AuthApi authApi = Wearable.getAuthApi(context); //인증 신청 시작

authApi.requestPermission("nodeId",Permission.DEVICE_MANAGER,Permission.NOTIFY)
            .addOnSuccessListener(새로운 OnSuccessListener<Permission[]>() {
                @보수
                public void onSuccess(Permission[] Permissions) { //권한 신청이 성공하여
                    권한 승인이 성공하여 권한을 반환합니다.
                }
            }).addOnFailureListener(새로운 OnFailureListener() { @Override public void

            onFailure(@NonNull Exception e) {
                    //허가 신청에 실패했습니다.
                }
            });
```

3. 장치 관리 및 상태 구독

3.1 장치 상태 쿼리 및 구독 지원

장치 관리 및 상태 구독 쿼리 결과		구독 이벤트 트리거 조건	구독 결과
연결 상태	1. 연결됨 2. 연결되지 않음	1. 휴대폰이 기기에 성공적으로 연결되었습니다. 2. 휴대폰과 기기의 연결이 끊어졌습니다.	1. 연결 성공 2. 연결 끊김 3. 연결 실패 4. 장치 삭제
배터리 상태	전력 값(0~100, 예: 98) 해당 없음		없음
충전 상태	1.충전 중 2.충전 안됨	1. 기기 충전 2. 완전 충전 3. 충전 중지	1. 충전 시작 2. 충전 완료 3. 충전 중지
착용상태	1. 착용 2. 착용하지 않음	1. 손목에 시계를 착용합니다. 2. 시계를 벗습니다.	1. 착용 2. 착용하지 않음
수면 상태	1. 수면 중 2. 깨어 있음	1. 시계를 차고 잠을 자세요. 2. 잠에서 깨어납니다.	1. 잠에 들다 2. 잠에서 깨어나다

3.2 상태 쿼리(Permission.DEVICE_MANAGER 권한을 신청해야 함)

```
//NodeApi 객체를 먼저 가져옵니다.
NodeApi api = Wearable.getNodeApi(context); //쿼리 메소드를 호출하
여 다양한 상태를 쿼리합니다. nodeId는 연결된 장치를 쿼리하여 얻은 장치 ID입니다. //현재 쿼리(연결 상태,
전원 상태, 충전 상태, 착용 상태)를 지원합니다. 상태, 절전 상태) api.query("nodeId",
DataItem.ITEM_CONNECTION)
        .addOnSuccessListener(new OnSuccessListener<DataQueryResult>() {
            @보수
            public void onSuccess(DataQueryResult result) { //쿼리 성공

                boolean connectionStatus = result.isConnected;//DataQueryResult가 연결됨을 확인합니다.
다양한 상태의 상태값은 DataItem에 하나씩 대응됩니다.
            }
        }).addOnFailureListener(새로운 OnFailureListener() { @Override public void

            onFailure(@NonNull Exception e) {
                //쿼리 실패
            }
    }); //
해당 웨어러블 디바이스 애플리케이션이 설치되어 있는
지 쿼리
    api.isWearAppInstalled("nodeId") .addOnSuccessListener(new OnSuccessListener<Boolean>() {
            @Override
            public void onSuccess(Boolean result) {
                //쿼리 성공. 애플리케이션이 설치되어 있으면 true를 반환하고, 애플리케이션이 설치되어 있지 않으면 false를 반환합니다.
            }
    }).addOnFailureListener(새로운 OnFailureListener() { @Override public void

            onFailure(@NonNull Exception e) {
                //쿼리 실패
            }
}); //
기기측 애플리케이션 열
기 //uri는 각 애플리케이션별로 맞춤설정되며 시계측 앱의 지정된 페이지를 여는 데
사용됩니다 api.launchWearApp("nodeId","uri")
```

```
        .addOnSuccessListener(새로운 OnSuccessListener<Void>() {
            @보수
            public void onSuccess(Void var1) { //웨어러블 장치 애플
                리케이션을 성공적으로 열었습니다.
            }
        }).addOnFailureListener(new OnFailureListener() { @Override public void

        onFailure(@NonNull @NotNull Exception var1) { //웨어러블 장치 애플리케이션을 열지 못했습니다.

        }
});
```

3.3 상태 구독(Permission.DEVICE_MANAGER 권한을 신청해야 함)

```
//NodeApi 객체를 먼저 가져옵니다.
NodeApi api = Wearable.getNodeApi(context); //데이터 변경 리스너를 생
성합니다.
onDataChangedListener = new OnDataChangedListener() {
        @보수
        공개 무효 onDataChanged(@NonNull 문자열 nodeId, @NonNull DataItem dataItem,
@NonNull DataSubscribeResult data) { //구독 상태 변경
            알림 수신 // 서로 다른 DataItem
            은 DataSubscribeResult의 서로 다른 상태에 해당하고, 하나는 if(dataItem.getType() ==
            DataItem.ITEM_CONNECTION.getType()){ 에 해당합니다.
                int connectionStatus = data.getConnectedStatus(); if(connectionStatus ==

DataSubscribeResult.RESULT_CONNECTION_CONNECTED){
                    //기기 연결 상태를 연결된 상태로 변경
                }
            }

        }

}; //다른 상태를 구독하려면 구독 메소드를 호출합니다. nodeId는 연결된 장치를 쿼리하여 얻은 장치 ID입니다. //현재 구독
을 지원합니다(연결 상태 변경, 충전 상태 변경, 착용 상태 변경, 절전 상태 변경) api .subscribe ("nodeId",
DataItem.ITEM_CONNECTION, onDataChangedListener) .addOnSuccessListener(new OnSuccessListener<Void>()
        { @Override public void onSuccess(Void var1) {


            //구독 추가 성공
        }
        }).addOnFailureListener(새로운 OnFailureListener() { @Override public void

            onFailure(@NonNull @NotNull 예외 var1) {
                //구독 추가 실패
            }
}); //
모니터링을 취소하려면 구독 취소 메소드를 호
출하세요. api.unsubscribe("nodeId", DataItem.ITEM_CONNECTION)
        .addOnSuccessListener(new OnSuccessListener<Void>() { @Override public void
            onSuccess(Void
            var1) {
                //구독 삭제 성공
            }
        }).addOnFailureListener(새로운 OnFailureListener() { @Override public void

            onFailure(@NonNull @NotNull 예외 var1) {
```

```
                //구독 삭제 실패
            }
});
```

4. 애플리케이션 간 메시지 통신(Permission.DEVICE_MANAGER 권한 신청 필요)

```
//데이터를 시뮬레이션합니다. 애플리케이션은 전송된 데이터
를 사용자 정의할 수 있습니다. byte[] messageBytes = new byte[1024] //
MessageApi 객체 가져오기 ;
MessageApi messageApi = Wearable.getMessageApi(context); //sendMessage 메서드
를 호출하여 웨어러블 장치 애플리케이션에 데이터를 보냅니다.
messageApi.sendMessage("nodeId",messageBytes) .addOnSuccessListener(new
        OnSuccessListener<Integer>() {
            @보수
            public void onSuccess(정수 결과) {
                //데이터 전송 성공
            }

        }) .addOnFailureListener(새로운 OnFailureListener() {
            @보수
            public void onFailure(@NonNull Exception e) { //데이터 전송 실패

            }
        });

//전송 리스너는 웨어러블 디바이스에서 전송되는 메시지를 모니터링하는 데 사용됩니다.
OnMessageReceivedListener onMessageReceivedListener = 새로 만들기
온메시지수신리스너() {
    @Override
    public void onMessageReceived(@NotNull 문자열 노드 ID, @NotNull byte[]
message) { //시계
            애플리케이션으로부터 메시지를 수신합니다.
    }

}; //웨어러블 장치 애플리케이션에서 보낸 메시지
를 수신합니다. messageApi.addListener("nodeId",
        onMessageReceivedListener) .addOnSuccessListener(new OnSuccessListener<Void>()
            { @Override
            public void onSuccess(Void var1) { //메시지 리스너 추가 성
                공
            }
        }).addOnFailureListener(새로운 OnFailureListener() { @Override public void

            onFailure(@NonNull @NotNull 예외 var1) {
                //메시지 수신 추가 실패
            }
}); //

messageApi.removeListener("nodeId",onMessageReceivedListener).addOnSuccessListener r(new OnSuccessListener<Void>()
{ @Override public void onSuccess(Void var1) {


            //메시지 모니터링을 성공적으로 취소했습니다.
    }
```

```
}).addOnFailureListener(새로운 OnFailureListener() {
        @보수
        public void onFailure(@NonNull @NotNull 예외 var1) {
                //메시지 모니터링 취소 실패
        }
});
```

5. 메시지 알림(Permission.NOTIFY 권한 필요)

```
//Get NotifyApi
NotifyApi informApi = Wearable.getNotifyApi(context); //메시지 알림을 보내려면
Permission.NOTIFY 권한이 있는지 미리 확인해야 합니다 .
informApi.sendNotify("nodeId","title","message") .addOnSuccessListener(new
                OnSuccessListener <상태>() {
                        @Override
                        public void onSuccess(상태 status) { if(status.isSuccess()){

                                        //알림이 성공적으로 전송되었습니다. 메시지 알림의 내용은 시계에 표시되지만 시계측 애플리케이션은 이를 인식하지 못합니다.
                                }
                        }
                }).addOnFailureListener(새로운 OnFailureListener() { @Override

        public void onFailure(@NonNull @NotNull Exception var1) { //알림 보내기 실패

        }
});
```

6. 서비스 연결 상태 관리 (권한 필요 없음)

```
//ServiceApi 가져오기
ServiceApi serviceApi = Wearable.getServiceApi(context); //이 리스너를 통해 타사 애플리
케이션과 Xiaomi Wear 앱 간의 연결 상태를 모니터링할 수 있습니다.
OnServiceConnectionListener onServiceConnectionListener = 새로 만들기
OnServiceConnectionListener() {
        @Override
        public void onServiceConnected() {
                //서비스 연결 성공
        }

        @Override
        public void onServiceDisconnected() {
                //서비스 연결이 끊겼습니다.
        }

}; //리스너 등록
serviceApi.registerServiceConnectionListener(onServiceConnectionListener); //리스너 취소

serviceApi.unregisterServiceConnectionListener(onServiceConnectionListener);
```