mazec for Business テクニカルガイド

外部アプリケーション連携（Android 版）

第 1 版

はじめに

本書では、法人向けmazec for Businessの概要や、外部アプリケーションと連携して使うための仕様について説明しています。

本書の記載内容は、Android開発言語であるJavaやXMLに関する基本知識を持っていることを前提としています。

コードの表記

- コードの記述例は、囲みの中に記述します。
- 1行に収まらない場合は、折り返すか、途中で改行して記述する場合があります。

```
<EditText ... android:privateImeOptions="input_mode=2&amp;filter=8" />
```

目次

第1章 法人向けmazecとは

法人向けmazecとは、コンシューマー向け製品とは異なり、以下の法人向け機能が用意されています。

- アプリケーション連携
- 異体字サポート
- タブレット UI
- 数字キーボード（テンキー）

アプリケーション連携

入力モードの切り替えや認識文字種の変更など、ユーザーが行うmazecの操作や設定の一部をお客様が開発するアプリケーションから行うことができます。

異体字サポート

異体字とは、標準の字体と同じ意味・発音を持つが、表記に差異がある漢字のことです。法人向けmazecでは、姓名でよく使われる異体字をサポートしています。

例えば、「渡辺」と手書きすると、かな漢字変換候補に「渡辺」「渡邉」「渡邊」と表示されます。

そのほかに「高田」「髙田」、「斉藤」「齋藤」「齊藤」などの異体字がサポートされます。

※機種依存により、異体字の表記には次の制約があります。

・異体字がほかの文字に変換される場合があります。(例：□、スペース、その他の文字など)
・異体字が標準の字体で表記される場合があります。（mazecは正しい文字コードを出力しています）

タブレット UI

７インチ以上の端末では法人向けmazec独自のタブレットUIで快適な操作ができます。

数字キーボード（テンキー）

こちらも法人向けmazec専用で、金額や電話番号などを効率良く入力することができます。

第2章 アプリケーション連携

2.1. mazec for Businessのアプリケーション外部連携概要

mazec for Businessは、次の2つのアプリケーションから制御することができます。

- Android OSネイティブアプリケーション
- ブラウザで動作するWebアプリケーション

Android OSネイティブアプリケーションからの制御

お客様が開発するAndroid OSネイティブアプリケーションからmazecの下記の動作を制御できます。

・入力モードの指定

・ 認識文字種の指定

・数字キーボード（テンキー）の配列やキーの非表示設定

以下の指定方法があります。

・入力コントロールごとに静的に指定する

・mazecの起動時に動的に指定する

ブラウザで動作するWebアプリケーションからの制御

ブラウザで動作するWebアプリケーションからmazecの下記の動作を制御できます。

・入力モードの指定

input要素のtype属性に応じて、mazecの入力モードが決定されます。

※ Web アプリケーションからは入力モード以外の制御は行えません。

2.2. mazec for Businessの機能

2.2.1 入力モードの切り替え

アプリケーションから入力モードを指定してmazecを起動することができます。

mazecによる入力方法には、次の４つのモードがあります。

- 手書き認識による交ぜ書き入力モード
- ソフトウェアキーボード入力モード
- 数字入力（テンキー）モード
- イメージ入力モード

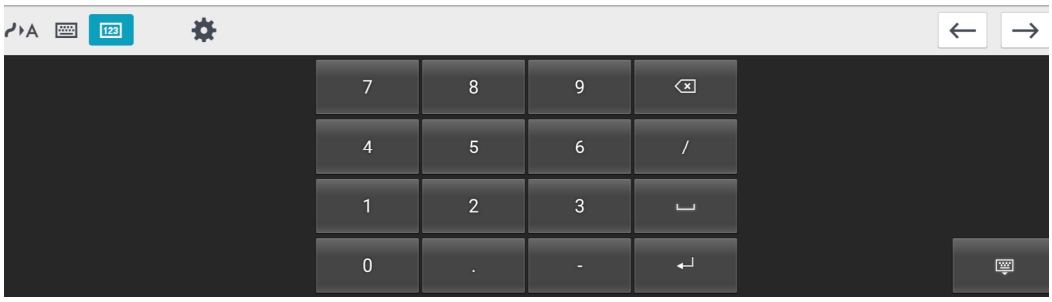
入力項目の内容に適した入力モードでmazecを起動（表示）できます。

例えば、住所や氏名の入力欄は手書きによる交ぜ書き入力モード、電話番号の入力欄には数字入力モード、署名の入力欄（自著欄）にはイメージ入力モードを指定することで、入力欄に応じた最適な入力方法を提供することができます。

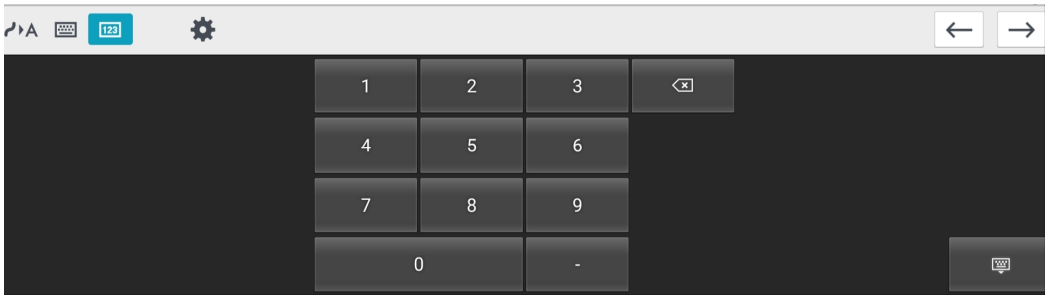
交ぜ書き入力モード

ソフトウェアキーボード入力モード（スタンダードモード／かな漢字変換ON）

「あいう/ABCキー」で、直接入力（半角英数字入力）とかな漢字変換を使った入力を切り替えて利用することができます。

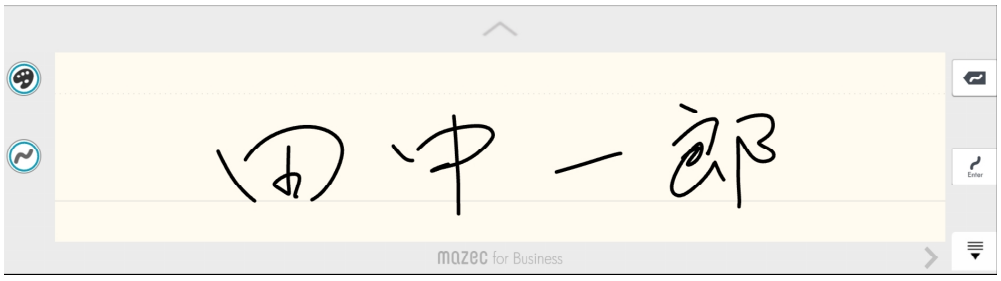


数字入力モード



数字入力モード（電話配列/一部のキーを非表示）

入力欄の目的に応じて不要なキーを非表示にすることができます。



イメージ入力モード

左バーにあるパレットボタンでストロークの色や太さを変更することができます。

2.2.2 認識文字種の設定

入力欄ごとに認識文字種を指定できます。特定の文字の種類を入力する場合、認識文字種を指定すると認識率が高くなります。
例えば、フリガナ用の入力欄に対して認識文字種をカタカナに設定すると、変換候補にはカタカナが優先的に表示されます。

入力モードと認識文字種を組み合わせて指定すると、より効率的な文字入力を実現できます。

認識文字種の設定（認識文字種：カタカナに設定）

2.3. プログラムインターフェースの仕様

2.3.1 入力コントロールへの入力モードと認識文字種の指定

入力コントロールに対して、入力モードや認識文字種を指定します。mazecは指定された入力モードと認識文字種で起動します。指定しない場合、以前の状態で表示されます。

例）

- 入力コントロールA：入力モード＝交ぜ書き、認識文字種＝漢字を指定
- 入力コントロールB：入力モード＝交ぜ書き、認識文字種＝ひらがなを指定

上記の場合、フォーカスを入力コントロールAから入力コントロールBに移動すると、入力モードが交ぜ書き、認識文字種がひらがなの状態でmazecが表示されます。
また、mazec以外のIMEからmazecに変更すると、指定した入力モードと認識文字種でmazecが表示されます。ただし、フォーカスを移動せずにmazecをいったん閉じて再表示した場合は、以前の状態で表示されます。

EditTextで指定する場合

例）layout.xml

```
<EditText ... android:privateImeOptions="input_mode=2&amp;filter=8" />
```

独自のビューで指定する場合

View.onCreateInputConnection をオーバーライドし、引数 EditorInfo のメンバ privateImeOptions で指定します。

例）

```
outAttrs.privateImeOptions = "input_mode=2&amp;filter=8";
```

input_mode / filterの値

プロパティ	値
input_mode	2：交ぜ書き 3：キーボード 4：イメージ入力 5：テンキー

プロパティ	値
filter	0：All 1：ひらがな 2：カタカナ 3：アルファベット 4：数字 7：記号 8：漢字 11：英数字

テンキーの種類/非表示キー

プロパティ	値
numpad_type	0：電卓配列 1：電話配列

プロパティ	値
numpad_hide_keys	13：Enter 32：スペース（空白） 45：ハイフン（マイナス） 46：ドット（少数点） 47：スラッシュ 複数指定する場合はカンマで区切ります

例）テンキー（電話配列、ハイフン以外のキーを非表示）を選択する場合

```
outAttrs.privateImeOptions =
          "input_mode=5&amp;numpad_type=1&amp;numpad_hide_keys=13,32,46,47";
```

2.3.2 mazecの起動時に入力モードと認識文字種を指定

アプリケーションからmazec（IME）を起動（表示）させるタイミングで、入力モードと認識文字種を指定することができます。IMEがmazecになっている状態で、入力コントロールにフォーカスがある場合に、指定したパラメータにしたがってmazecを起動させることが可能です。

InputMethodManager クラスの下記メソッドを使用します。

※詳しくは、Android Developer リファレンスを参照ください。

使用するメソッド

```
public void sendAppPrivateCommand (View view, String action, Bundle data)
```

入力モードの変更

- **アクション名**

```
com.metamoji.mazec.action.set_input_mode
```

- **データ**

キー名	型	値
data	int	2：交ぜ書き 3：キーボード 4：イメージ入力 5：テンキー

認識文字種の変更

- **アクション名**

```
com.metamoji.mazec.action.set_filter
```

- **データ**

キー名	型	値
data	int	0：All 1：ひらがな 2：カタカナ 3：アルファベット 4：数字 7：記号 8：漢字 11：英数字

2.3.3 input要素type属性による入力モードの切り替え

標準ブラウザのinput要素のtype属性の指定に応じて、mazecの入力モードが変わります。例えば、数字だけを入力する1行入力フィールドに対し、type属性に"number"を指定すると、mazecの数字キーボードを表示させることができます。

input要素のtype属性の指定によるmazecの入力モードは以下のようになります。

type属性	mazecの入力モード
text	交ぜ書き
search	交ぜ書き
tel	数字キーボード
url	交ぜ書き
number	数字キーボード
password	ソフトウェアキーボード
email	ソフトウェアキーボード

制限事項）

端末によっては、input要素のtype属性に"email"を指定したとき、mazecの入力モードがソフトウェアキーボードではなく、交ぜ書きになる場合があります。

2.3.4 イメージデータの入力

出力するイメージサイズの指定

input_modeの値に"4"が指定されている場合、mazecはイメージ入力モードで起動します。その際mazecは以下のメソッドを使用して出力するイメージのサイズを問いあわせます。

```
inputConnection.performPrivateCommand(String action, Bundle data)
```

アプリケーション側ではフレームワークから次のメソッドが呼び出されます。

```
public boolean onPrivateIMECommand (String action, Bundle data)
```

- **アクション名**

```
com.metamoji.mazec.action.query_image_size
```

- **データ**

キー名	型	値
data	null	–

出力イメージのサイズを指定（返答）するには、次のメソッドを使用します。

```
public void sendAppPrivateCommand (View view, String action, Bundle data)
```

● **アクション名**

```
com.metamoji.mazec.action.reply_image_size
```

● **データ**

キー名	型	値
width height	int int	画像の幅（ピクセル） 画像の高さ（ピクセル）

※ 出力イメージのサイズが指定されていない場合は、手書き入力領域の大きさに正規化して出力します。

※ タイミングによっては正しく設定されない場合があるため、出力サイズの指定（返答）はmazecからの問い合わせがあってから行ってください。

データの出力

イメージ入力モードで確定された場合、以下のメソッドを使用して、通常のテキストではなく入力されたストロークイメージをBitmapオブジェクトとしてアプリケーション側に通知します。

```
inputConnection.performPrivateCommand(String action, Bundle data)
```

アプリケーション側ではフレームワークから次のメソッドが呼び出されます。

```
public boolean onPrivateIMECommand (String action, Bundle data)
```

● **アクション名**

```
com.metamoji.mazec.action.commit_image
```

● **データ**

キー名	型	値
bitmap	Bitmap	画像

アプリケーションは、受け取った Bitmap オブジェクトを利用して、画面に表示したりファイルに保存したりできます。

サンプルコード

本サンプルコードは処理の概念を例示したものであり、アプリケーションの動作保証をするものではありません。

JavaやAndroid開発環境（SDK）のバージョンによっては修正が必要になる場合がありますので、Android Developerリファレンスにしたがってお客様の責務で開発をお願い致します。

Appendix A

EditTextのprivateImeOptionsで入力コントロールに入力モードと認識文字種を静的に指定するサンプルです。

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:ems="10"
        android:inputType="textMultiLine"
        android:privateImeOptions="input_mode=2&amp;filter=0">
    </EditText>

</RelativeLayout>
```

Appendix B

アプリケーション側から mazec にイメージ入力モードを指定し、mazec から出力されたイメージを表示するサンプルです。

- EditText 派生のテキストボックスをタップするとイメージ入力モードで mazec が起動します。
- 出力イメージのサイズは ６４０ × ２４０（px）を指定します。
- 出力されたイメージをイメージビューに描画します。

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:color/white">
    <com.metamoji.mazec.signature.test.MyEditText
        android:id="@+id/myEditText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="タップして画像入力"
        android:singleLine="true"
        android:privateImeOptions="input_mode=4"/>
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/white"
        android:onClick="buttonclick" />
</LinearLayout>
```

MyActivity.java

```
public class MyActivity extends Activity {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final MyEditText myEditText = (MyEditText)findViewById(R.id.myEditText);
        final ImageView imageView = (ImageView)findViewById(R.id.imageView);

        myEditText.setCommitImageListener(new MyEditText.CommitImageListener() {
            public void commitImage(Bitmap image) {
                imageView.setScaleType(ImageView.ScaleType.CENTER);
                imageView.setImageBitmap(image);
                imageView.invalidate();
            }
        });
    }
}
```

MyEditText.java

```
public class MyEditText extends EditText {
    public boolean onPrivateIMECommand (String action, Bundle data) {
        boolean handled = false;

        if (action.equals("com.metamoji.mazec.action.commit_image")) {
            // mazec から出力されたイメージ（Bitmap オブジェクト）を受け取る
            Bitmap bitmap = (Bitmap)data.get("bitmap");

            if (null != m_commitImageListener) {
                m_commitImageListener.commitImage(bitmap);
            }

            handled = true;
        } else if (action.equals("com.metamoji.mazec.action.query_image_size")) {
            // mazec から出力イメージサイズの問い合わせ
            InputMethodManager imm = (InputMethodManager)getContext().
                                getSystemService(Context.INPUT_METHOD_SERVICE);

            if (null != imm) {
                Bundle bundle = new Bundle();

                // 出力イメージのサイズを返答
                bundle.putInt("width", 640);
                bundle.putInt("height", 240);
                imm.sendAppPrivateCommand(this,
                    "com.metamoji.mazec.action.reply_image_size", bundle);

                handled = true;
            }
        }

        return handled ? true : super.onPrivateIMECommand(action, data);
    }

    public static interface CommitImageListener {
        public void commitImage(Bitmap image);
    }

    private CommitImageListener m_commitImageListener;

    public void setCommitImageListener(CommitImageListener listener) {
        m_commitImageListener = listener;
    }
}
```

－以上－