How to use BaseActivity

Step 1: Extends the BaseActivity class for the Activity used in application.
Step 2: Set view for the activity by setContentView method
Step 3: Implement pre-set methods below

- onBaseCreate: the method will be called after Android onCreate. Any new global objects need to be initiated should be here (see example)

```
private People people;

@Override
public void onBaseCreate() {
    people = new People("Tom", 25);
    // initiate global object "people" with name is Tom and age is 25
}
```

- onDeepLink: the method will be called after onBaseCreate to handle deeplink scenarios. The UIs objects (textview, button, etc) have not been binded at this point so no UIs invoking (see example)

```
@Override
public void onDeepLinking(Intent data) {
    if(data.getScheme().equals("my.application.scheme")) {
        String deeplink = data.getDataString();
        // handle deeplink data here
        // no UIs invoking - the UIs have not been binded yet
    }
}
```

- onNotification: the method will be called after onBaseCreate and onDeeplink to handle notification. The method will be called once per notification. The UIs objects (textview, button, etc) have not been binded at this point so no UIs invoking (see example)

```
@Override
public void onNotification(Intent data) {
    if(data.getBooleanExtra(Constant.NOTIFICATION_DEFINED, false)) {
        String message = data.getStringExtra("NOTIFICATION_MESSAGE");
        // handle message from notification
        // no UIs invoking - the UIs have not been binded yet
    }
}
```

- onBindView: the method will be called after the view of the activity is created and only called once when the activity is created. All the binding view must be here (see example).

```
private TextView demo_text_view;
private ListView demo_list_view;
private EditText demo_edit_text;

@Override
public void onBindView() {
    // the binding views should be in this method
    // check BaseActivity.findViewById for more details

    findViewById(R.id.demo_button);
    // if the object "demo_button" is not needed for the development,
    // only need to call findViewById, no cast action needed.
    // The action click will be handled in onSingleClick method

    demo_text_view = (TextView) findViewById(R.id.demo_text_iew);
    // In default, the text view will also receive click action and will
    // be handled in onSingleClick method

    demo_edit_text = (EditText) findViewById(R.id.demo_edit_text);
    // In default, edit text will not receive click action

    demo_list_view = (ListView) findViewById(R.id.demo_list_view);
    // In default, list view will not receive click action

    // check BaseInterface.isExceptionalView for more details

}
```

- onInitializeViewData: the method will be called after onBindView and is to setup the data to the binded views or calling webservice request for the first time entering the screen. The method will be called once when the activity is created (see example).

```
@Override
public void onInitializeViewData() {

    demo_text_view.setText("Initialized text");

    demo_edit_text.setHint("Initialized hint");

    // set up data for the view after binding

    makeRequest(TAG, true, new MyParam(), this,
            Constant.RequestTarget.WEBSERVICE_REQUEST, new String[]{});

    // make webservice request if required

    // all the initialized data to view must be here
}
```

- onBaseResume: the method is called alike to Android onResume but with some simple checking before (no need to worry about the checking)

- onBaseFree: the method is called when the activity/fragment is finished. Any view/data that needed to be unbinded/removed/invalidated should be handled here.

- onSingleClick: the method is integrated with the findViewById of the BaseActivity. All the views (not the exceptional views) will be automatically implemented the click-action listener and (optional) touch-action listener. The click-action will be handled here (see example).

```
@Override
public void onSingleClick(View v) {
    switch (v.getId()) {
        case R.id.demo_button:
            // action click of demo_button will be handled here
            // even though it didn't implement any on click listener
            break;
        case R.id.demo_text_iew:
            // same for the demo_text_view
            break;
        // the demo_edit_text and demo_list_view are exceptional
        // so those will not have the click action handled
    }
}
```

- showAlertDialog: the method will show a default (system style) alert dialog with customized title, message and confirm button. It also provides the listener for confirm action.

- showLoadingDialog: the method will show a default loading dialog and dimming the current screen that call the method.

- showDecisionDialog: the method will show a default confirmation dialog with 2 styles (2 buttons [YES, NO] and 3 buttons [YES, NO, CANCEL]). It also provides the listener for confirm actions.

- DONE

-

How to use BaseMultipleFragmentActivity and BaseMultipleFragment

- For the basic methods, the usage is described in “How to use BaseActivity” section.

- onInitializeFragments: the method will because once when the activity is created. The first fragment of the activity should be added here.

- onLastFragmentBack: the method will be invoked when user press back and the fragment stack size is one (the current fragment is the last fragment of the stack) so that the action will be custom handled (ie. finish/display exit dialog).

- addFragment: the method is to add another BaseFragment instance onto the stack of the current fragment container. The fragment will be identified by “tag” parameter, each fragment MUST have a unique tag. If adding an already exist in stack fragment, all the fragments will be removed to the added fragment.

- replaceFragment: the method is to replace the current fragment with another fragment. There are 2 modes defined by “clearStack” parameter. If “clearStack” is true, the replacing fragment will be the first fragment of the stack (all the current stack will be cleared), otherwise, the last fragment of the current stack will be replaced by replacing fragment.

- finish: the method is alike to the Android finish method.

- DONE

How-to make a webservice request

Step 1: Define webservice url

- Go to application.package.name.util.Constant class
- Modify Constant .SERVER_URL to the server host
- Modify timeout and retry policy (optional)
- Add the request target to RequestTarget enum
- Implement the build method of the RequestTarget enum (see example)

```
public enum RequestTarget {
    GET_EXAMPLE, POST_EXAMPLE, WEBSERVICE_REQUEST, BACKGROUND_REQUEST;

    public static String build(RequestTarget target, String... extras) {
        switch (target) {
            case GET_EXAMPLE:
                // suppose extras is the array (id, directory) of ["006", "image"]
                return String.format("/get_example/%s/%s").toString();
                // the url would automatically be http://serverhost.com/get_example/006/image
            case POST_EXAMPLE:
                return "/post_example";
                // the url would automatically be http://serverhost.com/post_example
            default:
                break;
        }
        return "";
    }
}
```

- DONE

Step 2: Define webservice

- Go to application.package.name.core.connection.Requester class
- Implement the case for handling this request in startWSRequest method (see example)

```
public static boolean startWSRequest(String tag, RequestTarget target,
                                     String[] extras, Param content, WebServiceResultHandler handler) {
    try {
        WebServiceRequest request;
        if (BaseProperties.wsRequester == null)
            BaseProperties.wsRequester = WebServiceRequester
                    .getInstance(CentralApplication.getContext());
        switch (target) {
            case GET_EXAMPLE:
                request = new WebServiceRequest(
                        tag, // the screen that makes the request
                        RequestType.HTTP, // http protocol
                        RequestMethod.GET, // GET method
                        Constant.SERVER_URL, // to the server host
                        target, // with target
                        RequestTarget.build(target, extras), // build url
                        content, // with parameters
                        BaseProperties.wsRequester, // ignore this
                        handler); // the handler for response result
                break;

            case POST_EXAMPLE:
                request = new WebServiceRequest(
                        tag, // the screen that makes the request
                        RequestType.HTTPS, // https protocol
                        RequestMethod.POST, // POST method
                        Constant.SERVER_URL, // to the server host
                        target, // with target
                        RequestTarget.build(target, extras), // build url
                        content, // with parameters
                        BaseProperties.wsRequester, // ignore this
                        handler); // the handler for response result
                break;
            default:
                throw new ActivityException(
                        "Requester: No request target found");
        }
        BaseProperties.wsRequester.startRequest(request);
        DLog.d(TAG, request.getRequestMethod().name().toUpperCase()
                + " >> " + request.getUrl());
        return true;
    } catch (Exception ex) {
        ex.printStackTrace();
        DLog.d(TAG, "Request canceled!");
        return false;
    }
}
```

- DONE

Step 3: Build request parameters

- Create the request parameter and extend the BaseParam class
- Implement the request parameter class (see example)

```
public class TestParam extends BaseParam {

    private String username;
    private String password;

    public TestParam(String username, String password) {
        this.username = username;
        this.password = password;
    }

    @Override
    public HashMap<String, String> makeRequestHeaders() {
        // build headers for the request here (if the request a specific headers)
        // if all requests required same headers, put it in the makeRequestHeaders of the BaseParam
        HashMap<String, String> extra_headers = super.makeRequestHeaders();
        extra_headers.put(Constant.Header.AUTHORIZATION.toString(), "Basic Authentication example");
        return extra_headers;
    }

    @Override
    public byte[] makeRequestBody() {
        // build request body here (only used for POST request)
        // for the GET request, build the parameters on the Constant.RequestTarget.build()

        JSONObject body = new JSONObject();
        try {
            // suppose the request body use JSON format
            body.put("username", username);
            body.put("password", password);

            return body.toString().getBytes();
        } catch (JSONException e) {
            e.printStackTrace();
        }


        return new byte[0];
    }
}
```

- DONE

Step 4: Build request result and parser

- Create the result parser for the request and extend BaseParser
- Implement the parser (see example)

```
public class TestParser extends BaseParser {

    @Override
    public BaseResult parseData(String content) {
        // the content is the response body of the request
        TestResult result = new TestResult();
        try {
            // parse the response body and convert to the result object
            JSONObject json = new JSONObject(content);
            String name = json.optString("name");
            int age = json.optInt("age");

            result.setName(name);
            result.setAge(age);

            // after finishing parsing without error, the result need to be set as OK
            result.setStatus(Constant.StatusCode.OK);
        } catch (JSONException e) {
            e.printStackTrace();
            result.setStatus(Constant.StatusCode.ERR_PARSING);
        }
        return result;
    }
}
```

- Go to application.package.name.core.base.BaseParser class
- Implement the case for handling this request in performParsing method (see example)

```
/**
 * This method perform parsing data
 */
private static BaseResult performParsing(String content,
                                         RequestTarget target) {
    BaseResult data = null;
    switch (target) {
        case POST_EXAMPLE:
            data = new TestParser().parseData(content);
        default:
            break;
    }

    return data;
}
```

- DONE

Step 5: Implement webservice result handler

- Make the activity that make the request to implement WebServiceResultHandler interface including onResultSuccess, onResultFail, onFail (see example)

```
public class TestActivity extends BaseActivity implements WebServiceRequester.WebServiceResultHandler {
    @Override
    public void onBaseCreate() {...}
    @Override
    public void onDeepLinking(Intent data) {...}
    @Override
    public void onNotification(Intent data) {...}
    @Override
    public void onBindView() {...}
    @Override
    public void onInitializeViewData() {...}
    @Override
    public void onBaseResume() {...}
    @Override
    public void onBaseFree() {...}
    @Override
    public void onSingleClick(View v) {...}
    @Override
    public void onResultSuccess(BaseResult result) {
        // handle the successful result
        if(result instanceof TestResult) {
            TestResult testResult = (TestResult) result;
            DLog.d("TAG", "Name: " + testResult.getName());
            DLog.d("TAG", "Age: " + testResult.getAge());
        }
    }

    @Override
    public void onResultFail(BaseResult result) {
        // handle the failed result
        if(result instanceof TestResult) {
            DLog.d("TAG", result.getMessage());
        }
    }

    @Override
    public void onFail(Constant.RequestTarget target, String error, Constant.StatusCode code) {
        // handle system / network errors
        DLog.d("TAG", "Request to " + target + " is failed due to " + error + " with the code of " + code);

    }
}
```

- DONE

Step 6: Put everything together

- On the screen that make the request, call method makeRequest and put everything together (see example)

```
@Override
public void onInitializeViewData() {
    makeRequest(
            "TestActivity", // tag of the screen making request
            true, // lock screen by loading dialog (false otherwise)
            new TestParam("example_username", "example_password"), // content parameters
            this, // the screen will handle the response result
            Constant.RequestTarget.POST_EXAMPLE, // send request to POST_EXAMPLE target
            new String[]{"006", "image"}); // with some extra parameters to build url
}
```

- DONE

-- END OF DOCUMENT --