Plugin Implementation Guide:
Sana Client version 1.1

Sana Development Team

January 18, 2013

Contents

Introduction

This document provides an overview and implementation guidelines for using 3rd party applications to collect data as part of a Sana procedure. It is not intended as a full reference resource for constructing Sana procedures.

1 Requirements and Prior Knowledge

This section describes the minimal requirements for working with the Sana client plugin architecture.

1.1 Hardware Requirements

Android based smart phone
PC with Android emulator

1.2 Software Requirements

Min Client Version	1.1
Min Android Version	1.6
*XML Editor with schema support(Optional)	

1.3 Prior Knowledge

Android application development
General XML document structure
*Sana XML Procedure format

2. Procedure Integration

2.1 Integrating an existing application

Starting with the simple Sana Procedure below, an external application can be integrated into Sana by adding the appropriate element to a Procedure using either the PLUGIN or PLUGIN ENTRY element type. Which you choose will depend on whether you are collecting data as a file, PLUGIN, or character sequence, PLUGIN ENTRY.

PLUGIN and PLUGIN ENTRY elements example

```
<Procedure title="Plugin Example" author="Sana"
      description="Demonstrates 1.x Plugin architecture">
      <!-- Plugin template
      <Page>
            <Element type="PLUGIN | ENTRY_PLUGIN" id="$VALUE"
                  concept="$VALUE"
                  question="$VALUE per concept"
                  action="[Intent action string]"
```

```
                        mimetype="$VALUE"/>
            </Page>
            action and mimetype attributes are required
            -->
            <Page>
                  <Element type="PLUGIN" id="01"
                        concept="SANA_PLUGIN_FILE"
                        question="Demonstrate plugin file capture"
                        action="org.sana.plugin.CAPTURE_FILE"
                        mimeType="application/xml"/>
            </Page>
            <Page>
            <Element type="ENTRY_PLUGIN" id="2"
                  concept="SANA_PLUGIN_TEXT"
                  question="Demonstrate plugin text capture"
                  action="org.sana.plugin.CAPTURE_CHARS"
                  mimeType="text/plain"/>
            </Page>
      </Procedure>
```

2.2 A word about attributes

The Sana application will default to try using the action attribute to launch your application. If the action attribute is not present, it will fall back to using the mimetype through the Android content resolution framework to try and discover an application registered to capture data of the specified mimetype. The former is the preferred method since the latter can produce unexpected results if multiple applications have been registered for the specified mimetype. NOTE: The standard naming convention is to have the action attribute defined as

application.package.concept

*w*here concept is the concept attribute in the xml with whitespace replaced by underscores.

If unsure whether an application provides an action string please consult the application documentation or consult with the application developers. The following section for developers describes how to properly construct an application to work seamlessly with Sana.

3. Developers. Building a custom application to work with Sana

This section provides instructions for building an application to integrate seamlessly with the plug-in architecture.

3.1 Android Application Manifest

While Sana can use implicit Intents to resolve and launch your application it is much more efficient to declare an action string in an intent-filter within your application’s manifest similar to the example below

```
      <activity android:name=".PluginStreamTest"
            android:label="@string/app_name" >
            <intent-filter>
```

```
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                  <action android:name="org.sana.plugin.CAPTURE_FILE" />
                  <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                  <action android:name="android.intent.action.VIEW" />
                  <category android:name="android.intent.category.DEFAULT" />
                  <data android:mimeType="application/xml" />
            </intent-filter>
      </activity>
      <activity android:name=".PluginCharTest"
            android:label="@string/app_name" >
            <intent-filter>
                  <action android:name="android.intent.action.DEFAULT" />
                  <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                  <action android:name="org.sana.plugin.CAPTURE_CHARS" />
                  <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                  <action android:name="android.intent.action.VIEW" />
                  <category android:name="android.intent.category.DEFAULT" />
                  <data android:mimeType="text/plain" />
            </intent-filter>
      </activity>
```

3.2 Additional state data passed to the plugin

The Intent used to launch the plugin Activity also includes the Encounter id, Patient identifier, an observation id-the Procedure Element id in the XMLL, and a concept name . This metadata can be retrieved from the launch Intent as follows, where observation, encounter, subject, and concept are all String values:

```
      observation = getIntent().getStringExtra("observation");
      encounter = getIntent().getStringExtra("encounter");
      subject = getIntent().getStringExtra("subject");
      concept = getIntent().getStringExtra("concept");
```

Note: The additional metadata is included as a way to uniquely identify captured data.

3.3 Returning data

Within the Activity that captures the data, the data can be returned to Sana by setting the Activity’s result to RESULT OK and returning any collected data and it’s type.

```
      Intent result = new Intent();
      result.setDataAndType(data, getIntent().getType());
      setResult(RESULT_OK, result);
```

3.3.1 Returning Files

Application should not attempt to pass binary data directly in the returned Intent. Instead, the application should return the Uri of the file where the data resides either a file or content-style Uri. In the case of a content-style Uri, your application should include a ContentProvider. Note: Reviewing the collected data will require that at least one Activity has an intent-filter declared for the data type.

3.3.2 Character Sequences

Character sequences should be returned as the fragment of a “content://text” style Uri and having a data type of “text/plain”.

```
String frag = ”Some character sequence data”;
Uri data = Uri.fromParts("content", "text", frag);
Intent result = new Intent();
intent.setDataAndType(data, "text/plain");
finish();
```

Note: Sana does not attempt to parse or otherwise validate the format of any character sequences returned by a plugin. Hence, your application should perform any validation prior to setting the result and returning.

3.3.3 Returning multiple observations

This is highly discouraged but possible with some modification to the client code. The general mechanism is similar to that employed by the PICTURE element where the answer is parsed immediately prior to transmission to create one or more response objects. Again, this technique is highly discouraged and unecessary in future releases due to changes in the procedure structure.

4. Additional resources

XML Schema for Sana procedures above and online validator
http://dev.sana.csail.mit.edu/xml

Sana wiki documentation for Procedures
http://sana.mit.edu/wiki/index.php?title=How to Define Your Own Procedures

Sana Android client download
http://sana.googlecode.com/files/release-1.1.apk

Sample plugin project with