CUP

Programming Guide

Version 1.0

Table of Contents

1. Overview

Companion UI Platform (CUP) allows you to display information from a host device, for example a smartphone, on companion devices and devices that use CUP.

CUP includes many control widgets (winsets), for example, button, label, list and spin, which allow you to create layouts to display information. The host device uses CUP to control the companion device's display.

The host and companion device connect and transfer data using Bluetooth. You can display the CUP winset on wearable devices (CUP Browser) of various resolution types, and get CUP winset user interaction events back from the wearable devices.

1.1. Basic Knowledge

The host device contains host applications that use CUP and contain the logical part of the CUP application. A host device is, for example, a smartphone using the Android platform with an application that integrates CUP to control a companion device. The host application requests a winset from the CUP Browser.

The companion device is a wearable device with a real-time-OS or another kind of OS, The companion device follows the protocol to communicate with the host device.

Figure 1: CUP overview

1.2. Architecture

The following figure shows the CUP architecture.

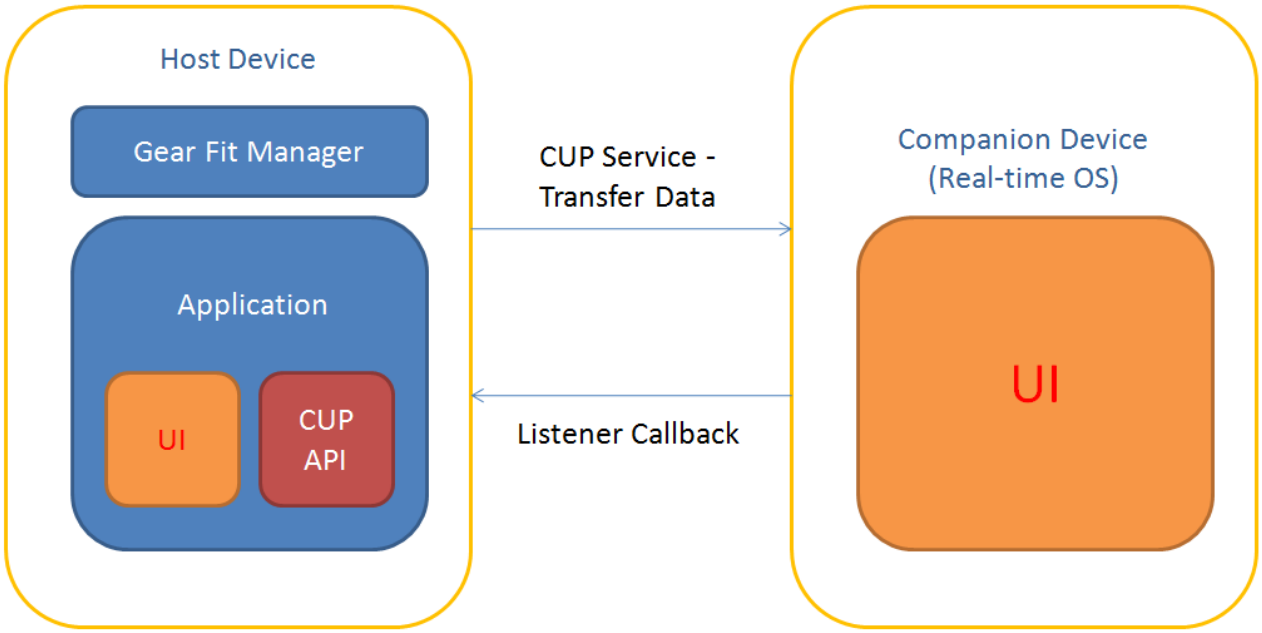

Figure 2: CUP architecture

The architecture consists of:

- **Applications:** One or more applications that use CUP.
- **CUP API:** Components for creating and showing various layouts on the companion device, including the callback interface.
- **CUP Service:** Service used to connect between the host and companion device, and transfer data between them.

1.3. Class Diagram

The following figure shows the CUP classes that you can use in your application.

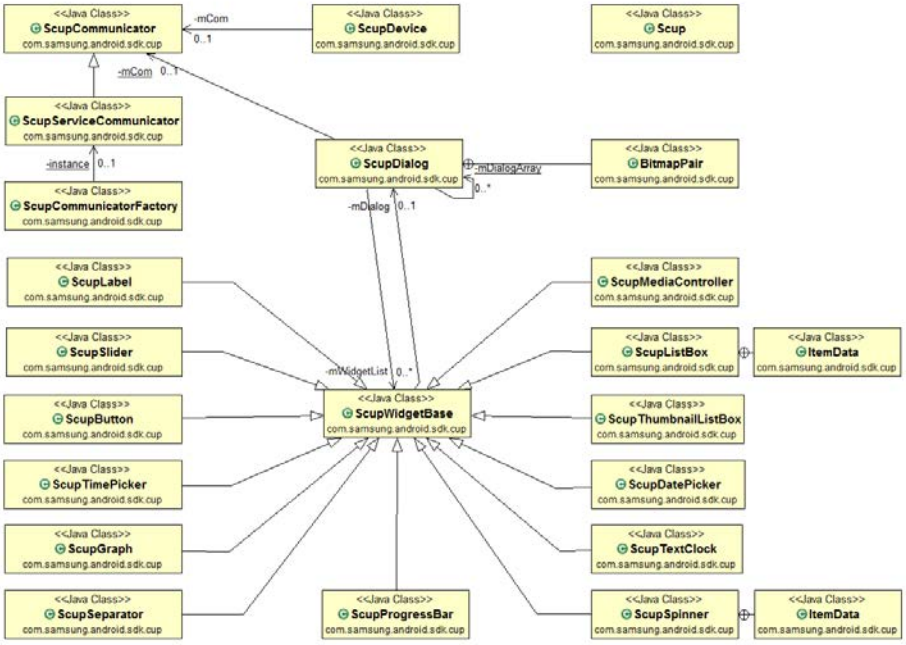

Figure 3: CUP classes

The CUP classes include:

- **Scup:** Initializes CUP.
- **ScupWidgetBase:** Provides basic widget functions in companion devices.
- **ScupButton:** Provides clickable buttons for the companion UI, with a callback listener for the button clicks.
- **ScupButton.ClickListener**: Triggers a callback when the user clicks the button on the companion device.
- **ScupDialog**: Supports displaying a screen on the companion device. Each dialog can contain multiple widgets.
- **ScupDialog.ActionButtonClickListener:** Triggers a callback when the user clicks the submit button in the dialog.
- **ScupDialog.TapListener:** Triggers a callback when the user taps the CUP device screen in the dialog.
- **ScupDialog.BackPressedListener:** Triggers a callback when the user clicks the back button in the dialog.
- **ScupLabel:** Shows information and images.
- **ScupListBox:** Shows a list of information.
- **ScupListBox.ItemListener:** Triggers a callback when the user selects a list item.
- **ScupSpinner:** Shows the same spinner as in Android.
- **ScupSpinner.ClickListener:** Triggers a callback when the user clicks a spinner item.

1.4. Supported Platforms

Android 4.2 (Ice Cream Sandwich API 17) or above supports CUP.

1.5. Supported Features

CUP supports the following features:

- Making basic components for Wingtip, for example, labels, buttons, spins, and lists
- Using listeners from Wingtip for, for example, tap, back press, and click events

1.6. Components

- Components
  - cup-v1.0.0.jar
  - sdk-v1.0.0.jar
- Imported packages:
  - com.samsung.android.sdk.cup

1.7. Installing the Package for Eclipse

To install CUP for Eclipse:

1. Add the cup-v1.0.0.jar file to the libs folder in Eclipse.

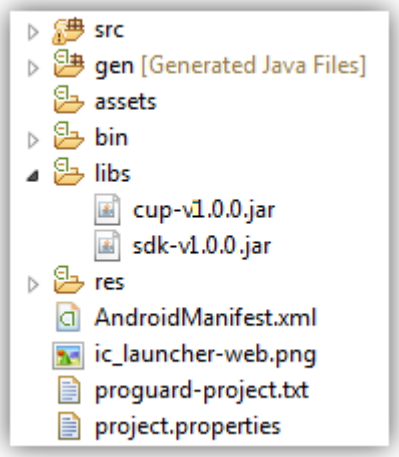

Figure 4: libs folder in Eclipse

2. Add the following permissions to your Android manifest file:

```
<uses-permission
      android:name="com.samsung.android.sdk.permission.SAMSUNG_CUP_SERVICE"/>
```

3. Add metadata to your Android manifest file:

```
<meta-data
       android:name="SAMSUNG_CUP_APP"
       android:value="APP_NAME;icon_name;true" />
```

4. To be able to transfer data with the CUP remote service, add the following tag into activity or service in your Android manifest file:

```
<intent-filter>
       <action android:name="com.samsung.android.sdk.cup" />
</intent-filter>
```

2. Hello CUP

To show a screen on a companion device, use ScupDialog or its subclass.

1. Create the first simple view in the CUP device.
2. Set up the parameters in the Manifest.xml file for running the program.

Hello CUP is a simple program that:

1. Shows the “Hello CUP” string on the Wingtip screen.
2. Has a Back button to finish the dialog.

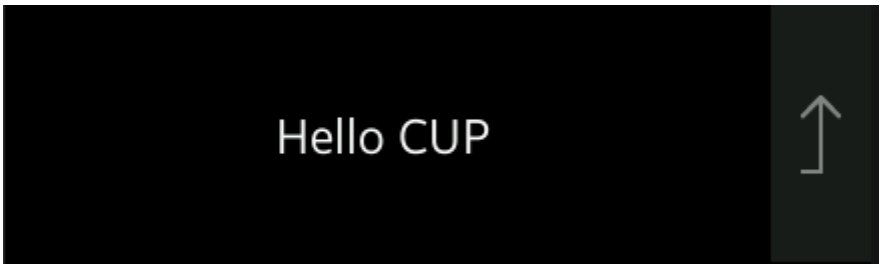

Figure 5: Hello CUP

```
public class HelloCupDialog extends ScupDialog {

	public HelloCupDialog(Context context) {
		super(context);
		// TODO Auto-generated constructor stub
	}

	@Override
	protected void onCreate() {
		// TODO Auto-generated method stub
		super.onCreate();

		setBackEnabled(true);

		ScupLabel helloLabel = new ScupLabel(this);
		helloLabel.setText("Hello CUP");
		helloLabel.setAlignment(ScupLabel.ALIGN_CENTER);
		helloLabel.setWidth(ScupLabel.FILL_DIALOG);
		helloLabel.setHeight(ScupLabel.FILL_DIALOG);
		helloLabel.show();

		setBackPressedListener(new BackPressedListener() {

			@Override
			public void onBackPressed(ScupDialog arg0) {
				// TODO Auto-generated method stub
				finish();
			}
		});
	}
}
```

MainActivity.java:

```
public class MainActivity extends Activity {

String[] NAMES = {"Hello Cup"};
private HelloCupDialog mHelloCupDialog = null;
// The item of list
private static final int Hello_Cup = 0;

@Override
	protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);

	 ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
	                    android.R.layout.simple_list_item_1, NAMES);
	mListView = (ListView) findViewById(R.id.demo_list);
	mListView.setAdapter(adapter);
	 mListView.setOnItemClickListener(new OnItemClickListener() {
		@Override
	public void onItemClick(AdapterView<?> parent, View view,
	                            int position, long id) {
	// S Pen SDK Demo programs
	if (position == Hello_Cup) {
		if (mHelloCupDialog == null) {
			mHelloCupDialog = new HelloCupDialog(
			getApplicationContext());
		} else {
			mHelloCupDialog.finish();
			mHelloCupDialog = null;
		}
	}
	});
	}
}
```

activity_main.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="Title"
        android:textColor="#000000"
        android:textSize="30sp"
        android:textStyle="bold" />

    <ListView
```

```
        android:id="@+id/demo_list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />

</LinearLayout>
```

AndroidManifest.xml configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.scupsamplev1"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission
android:name="com.samsung.android.sdk.permission.SAMSUNG_CUP_SERVICE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.scupsamplev1.MainActivity"
            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="com.samsung.android.sdk.cup" />
            </intent-filter>
        </activity>

        <meta-data
            android:name="SAMSUNG_CUP_APP"
            android:value="HELLOCUP;ic_launcher;true" />
    </application>

</manifest>
```

3. Using the Scup Class

The Scup class provides the following methods:

- `initialize()` initializes CUP. You need to initialize the CUP package before you can use it. If the device does not support CUP, SsdkUnsupportedException is thrown.
- `getVersionCode()` gets the CUP version number as an integer.
- `getVersionName()` gets the CUP version name as a string.

```
Scup scup = new Scup();
try {
	// initialize an instance of Scup
	scup.initialize(getApplicationContext());
} catch (SsdkUnsupportedException e) {
	if (e.getType() == SsdkUnsupportedException.VENDOR_NOT_SUPPORTED) {
		// Vendor is not Samsung.
	}
}

int versionCode = scup.getVersionCode();
String versionName = scup.getVersionName();
```

3.1. Using the initialize() Method

The `Scup.initialize()` method:

- Initializes the CUP package.
- Checks if the device is a Samsung device.

```
void initialize(Context context) throws SsdkUnsupportedException
```

If the CUP package fails to initialize, the `initialize()` method throws an SsdkUnsupportedException exception. To find out the reason for the exception, check the exception message.

3.2. Handling SsdkUnsupportedException

If an SsdkUnsupportedException exception is thrown, check the exception message type using `SsdkUnsupportedException.getType()`.

The following type of exception message is defined in the Scup class:

- **VENDOR_NOT_SUPPORTED:** The device is not a Samsung device.

4. Using the CUP Package

The following sections describe how to use the CUP package. You must first create a dialogue, and then you can add other widgets to display them in the dialog.

4.1. Creating and Using Dialogs

To show a screen on the companion device, use ScupDialog or its subclass:

1. Create the dialog.
2. Define the dialog life-cycle and implement the dialog callback.
3. Add a title bar and a listener for button clicks, if needed.
4. Change the dialog properties.
5. Add widgets for the dialog to show.
6. Implement ScupDialog.GestureListener to listen for various gesture events, if needed.
7. Implement ScupDialog.ActionButtonClickListener to listen for action button clicks, if needed.
8. Implement ScupDialog BackPressedListener to listen to back button clicks, if needed.

4.1.1. Dialog Sample Application

The following figure and code examples illustrate how you can create a CUP dialog sample application with the ScupDialog class and how the dialog looks on the CUP device screen.

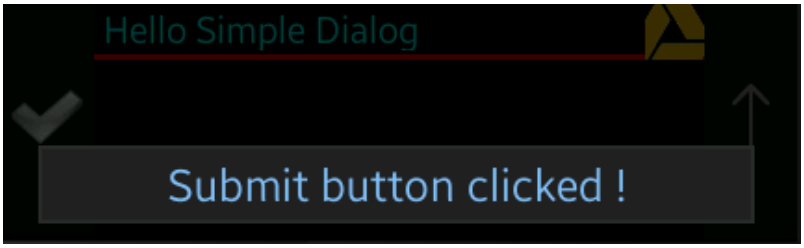

Figure 6: ScupDialog

```
public class SampleDialog extends ScupDialog {

	private static final String TAG = "DialogSample";

	// Constructor
	public SampleDialog(Context context) {
		// Need to call constructor of base class.
		super(context);
	}

	// After finish create,
	@Override
	protected void onCreate() {
		super.onCreate();
```

```
        // Add attribute of a dialog if user need

        // Set alignment and gap between controls
        this.setWidgetGap(2F);
        this.setWidgetAlignment(ScupDialog.WIDGET_ALIGN_HORIZONTAL_CENTER);

        // Enable the back button in the right side of companion screen
        // If false, the back button cannot display
        this.setBackEnabled(true);

        this.setBackPressedListener(new BackPressedListener() {

            @Override
            public void onBackPressed(ScupDialog dialog) {
                SampleDialog.this.showToast("Back button clicked !",
                        ScupDialog.TOAST_DURATION_SHORT);

                if (mBackButtonListener != null) {
                    mBackButtonListener.onBackButtonPressed(dialog);
                }

                SampleDialog.this.finish();
            }
        });

        // Background color
        this.setBackgroundColor(Color.BLACK);

        // Set title for title bar
        // Text color and text size
        this.setTitle("Hello Simple Dialog");
        this.setTitleTextColor(Color.CYAN);
        this.setTitleTextSize(6);

        // Border style of title
        this.setTitleBackgroundColor(Color.RED);
        this.setTitleBorderStyle(ScupDialog.TITLE_BORDER_UNDERLINE);
        this.setTitleBorderColor(Color.RED);
        this.setTitleBorderWidth(0.5f);

        this.setTitleButton(R.drawable.ic_launcher);
        this.setTitleButtonClickListener(new TitleButtonClickListener() {

            @Override
            public void onClick(ScupDialog arg0) {
                SampleDialog.this.showToast("Title button clicked !",
                        ScupDialog.TOAST_DURATION_SHORT);

                SampleDialog.this.finish();
            }
        });

        this.setActionButtonClickListener(new ActionButtonClickListener() {

            @Override
            public void onClick(ScupDialog arg0) {
                SampleDialog.this.showToast("Submit button clicked !",
                        ScupDialog.TOAST_DURATION_SHORT);
            }
```

```
        });

        this.setGestureListener(new GestureListener() {

            @Override
            public void onSingleTap(ScupDialog arg0, float x, float y) {
                // TODO Auto-generated method stub
                SampleDialog.this.showToast(
                        String.format("Tap at : (%f, %f)", x, y),
                        ScupDialog.TOAST_DURATION_SHORT);
            }

            @Override
            public void onFlick(ScupDialog arg0, int arg1, int arg2) {
                // TODO Auto-generated method stub

            }

            @Override
            public void onDoubleTap(ScupDialog arg0, float arg1, float arg2) {
                // TODO Auto-generated method stub

            }
        });

        // Enable the back button in the right side of companion screen
        // If false, the back button cannot display
        this.setBackEnabled(true);

        // background color
        this.setBackgroundColor(Color.BLACK);

        this.setActionButton(R.drawable.button_ok);
    }

    @Override
    protected void onResume() {
        Log.d(TAG, "onResume Called");
        super.onResume();
    }

    @Override
    protected void onPause() {
        Log.d(TAG, "onPause Called");
        super.onPause();
    }

    @Override
    protected void onDestroy() {
        // callback function if need
        if (mOnDestroyListener != null) {
            mOnDestroyListener.onDestroy();
        }

        super.onDestroy();
    }

    interface BackButtonListener {
        public void onBackButtonPressed(ScupDialog dialog);
```

```
	}

	private OnDestroyListener mOnDestroyListener;
	private BackButtonListener mBackButtonListener;

	interface OnDestroyListener {
		public void onDestroy();
	}
}
```

In MainActivity, the HelloCupDialog application uses the SampleDialog class defined above. Note that the MainActivity.java file is set as shown below for all the code examples in this document.

```
public class MainActivity extends Activity {

String[] NAMES = {"Hello Cup", "Dialog Widget Sample"};
private HelloCupDialog mHelloCupDialog = null;
private SampleDialog mSampleDialog;
// The item of list
private static final int Hello_Cup = 0;
private static final int Dialog_Widget_Sample = 1;
@Override
	protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);

	 ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
	                    android.R.layout.simple_list_item_1, NAMES);
	mListView = (ListView) findViewById(R.id.demo_list);
	mListView.setAdapter(adapter);
	 mListView.setOnItemClickListener(new OnItemClickListener() {
		@Override
	public void onItemClick(AdapterView<?> parent, View view,
	                        int position, long id) {
	// S Pen SDK Demo programs
	if (position == Hello_Cup) {
		if (mHelloCupDialog == null) {
			mHelloCupDialog = new HelloCupDialog(
			getApplicationContext());
		} else {
			mHelloCupDialog.finish();
			mHelloCupDialog = null;
		}
	}
	 else if (position == Dialog_Widget_Sample) {
		if (mSampleDialog == null) {
			mSampleDialog = new SampleDialog(
			getApplicationContext());
		} else {
			mSampleDialog.finish();
			mSampleDialog = null;
		}
	}
	});
	}
}
```

4.1.2. Creating the Dialog and Defining its Life-cycle

To display a screen on a companion device, create a ScupDialog subclass and set its properties. You need to know the dialog life-cycle so you can handle the processing. Implement the dialog constructor and call the constructor base class.

```
	// Constructor
	public SampleDialog(Context context) {
		// Need to call constructor of base class.
		super(context);
	}
```

CUP supports the following dialog callbacks:

- `OnCreate()`: Called after the dialog is successfully instantiated. You can start adding widgets and setting dialog attributes.
- `OnResume()`: Called when the dialog is ready to show on the companion device.
- `OnPause()`: Called when the dialog is still partially visible, but the user cannot interact with it.
- `OnDestroy()`: Called when `dialog.finish()` is called to destroy the dialog. This dialog is removed from the companion device.

The following figure illustrates the ScupDialog life-cycle.

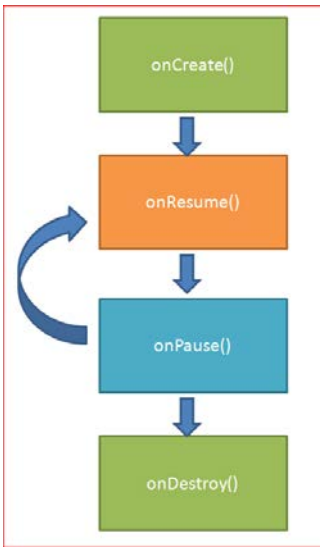

Figure 7: ScupDialog life-cycle

Create a subclass of ScupDialog that is shown on the companion device, and implement the needed life-cycle methods, for example, the constructor, `OnCreate()`, or `OnDestroy()`.

```
public class SampleDialog extends ScupDialog {

	// Constructor
	public SampleDialog(Context context) {
		// Need to call constructor of base class.
```

```
        super(context);
    }

    // After finish create,
    @Override
    protected void onCreate() {
        super.onCreate();

        // Add attribute of a dialog if user need

        // Enable the back button in the right side of companion screen
        // If false, the back button cannot display
        setBackEnabled(true);
        // background color
        setBackgroundColor(Color.BLACK);

        // Set title for title bar
        setTitle("Hello Simple Dialog");
    }

    @Override
    protected void onResume() {
        super.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
    }

    @Override
    protected void onDestroy() {
        // Callback function if need
        if (mOnDestroyListener != null) {
            mOnDestroyListener.onDestroy();
        }

        super.onDestroy();
    }
```

4.1.3. Adding the Title Bar

The title is the top layout of the companion screen. You can use the title, if you want to show something on the top of the screen. If you use the title, the content layout is smaller than without the title.

The title can show text information and a small button image. You can add a listener for the title button clicks.

```
        // Set title for title bar
        // Text color and text size
        setTitle("Hello Simple Dialog");
        setTitleTextColor(Color.CYAN);
        setTitleTextSize(6);
```

```
// Border style of title
setTitleBorderStyle(ScupDialog.TITLE_BORDER_UNDERLINE);
setTitleBorderColor(Color.RED);
setTitleBorderWidth(0.5f);

setTitleButton(R.drawable.sample);
setTitleButtonClickListener(new TitleButtonClickListener() {

      @Override
      public void onClick(ScupDialog arg0) {
            SampleDialog.this.showToast("Title button clicked !",
                        ScupDialog.TOAST_DURATION_SHORT);

            SampleDialog.this.finish();
      }
});
```

4.1.4. Changing the Dialog Properties

You can change the properties of the dialog to create a variety of UI screens. You can change the following dialog properties:

- Back button visibility
- Background color or image
- Animation type
- Widget gap between widgets
- Submit button visibility

```
// Enable the back button in the right side of companion screen
// If false, the back button cannot display
setBackEnabled(true);

// Background color
setBackgroundColor(Color.BLACK);

setActionButton(R.drawable.button_ok);

// Change animation type to ANIMATION_ALPHA
setAnimationType(ScupDialog.ANIMATION_ALPHA);
```

Note: The `setAnimationType(int type)` method shows the dialog with an animation, such as ANIMATION_ALPHA, ANIMATION_NONE, ANIMATION_ROTATE_X, ANIMATION_SCALE, ANIMATION_TRANSLATE, or ANIMATION_TRANSLATE_REVERSE.

4.1.5. Adding Widgets

After you have successfully created the dialog and the `onCreate()` callback is called, you can add widgets to display in the dialog.

CUP supports various widgets. For example, you can show in your dialog the following:

- Labels: The ScupLabel class displays a label with text and image. The label widget only has a gesture listener.
- Buttons: The ScupButton class shows a button on the side of the dialog, with icon and text. The button widget has a listener for the button clicks.
- Lists: The ScupListBox class shows a list of items of the same format. The user can select from different ListBox formats.
- Spinners: The ScupSpinner class allows the user to select items with a simple UI. The user can navigate between items by clicking the widget's up and down arrows.

Note: Widgets are aligned in the dialog with the following attributes of the ScupDialog class: WIDGET_ALIGN_HORIZONTAL_CENTER, WIDGET_ALIGN_NONE, and WIDGET_ALIGN_VERTICAL_CENTER. By default, widgets are displayed from the left of the screen.

4.1.6. Implementing the Gesture Listener

The ScupDialog.GestureListener interface detects actions when the user uses the companion device. GestureListener detects single tap, double tap, and flick events on the companion device.

```
this.setGestureListener(new GestureListener() {

	@Override
	public void onSingleTap(ScupDialog arg0, float x, float y) {
	// TODO Auto-generated method stub
		SampleDialog.this.showToast(
		String.format("Tap at : (%f, %f)", x, y),
		ScupDialog.TOAST_DURATION_SHORT);
	}

	@Override
	public void onFlick(ScupDialog arg0, int arg1, int arg2) {
	// TODO Auto-generated method stub

	}

	@Override
	public void onDoubleTap(ScupDialog arg0, float arg1, float arg2) {
	// TODO Auto-generated method stub

	}
});
```

4.1.7. Implementing the Action Button Click Listener

Implement the ScupDialog.ActionButtonClickListener interface to know when the user clicks the submit button in the dialog.

```
	setActionButton (R.drawable.button_ok);
	this.setActionButtonClickListener(new ActionButtonClickListener() {
		@Override
		public void onClick(ScupDialog arg0) {
			SampleDialog.this.showToast("Submit button clicked !",
					ScupDialog.TOAST_DURATION_SHORT);
			}
```

```
        });
```

4.1.8. Implementing the Back Button Listener

When you enable the back button, add the ScupDialog.BackPressedListener interface to listen for the back button click events.

```
        // Enable the back button in the right side of companion screen
        // If false, the back button cannot display
        setBackEnabled(true);

        setBackPressedListener(new BackPressedListener() {

                @Override
                public void onBackPressed(ScupDialog dialog) {
                        SampleDialog.this.showToast("Back button clicked !",
                                        ScupDialog.TOAST_DURATION_SHORT);

                        if (mBackButtonListener != null) {
                                mBackButtonListener.onBackButtonPressed(dialog);
                        }

                        SampleDialog.this.finish();
                }
        });
```

4.2. Using Buttons

The ScupButton class represents a button widget. To use the button:

1. Create a subclass of the ScupDialog class.
2. Add the ScupButton widget to the application.

The following figure and code examples illustrate the ScupButtonSampleApp application that creates a ScupButton instance in the dialog with some button-specific attributes.

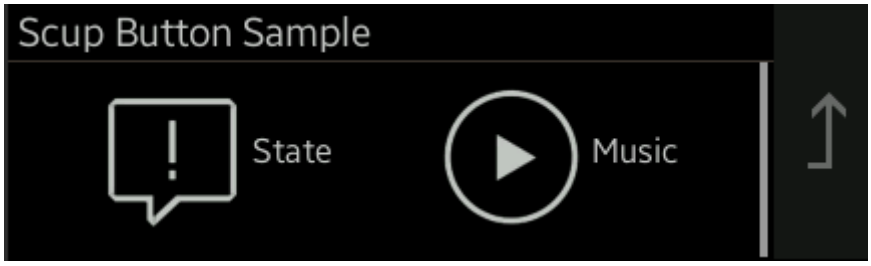

Figure 8: ScupButton

```
public class ScupButtonSampleApp extends ScupDialog {

        int[] color = {Color.BLUE,Color.BLACK};
        int i=0;
        public ScupButtonSampleApp(Context context) {
                super(context);
```

```
	}

	@Override
	protected void onCreate() {
		super.onCreate();
		this.setWidgetAlignment(ScupDialog.WIDGET_ALIGN_HORIZONTAL_CENTER);
		// Update the view in each status
		buttonChange();
		// Animation of show
		setAnimationType(ANIMATION_ALPHA);
	}

	public void buttonChange() {
		this.setTitle("Scup Button Sample ");
		// Button state
		final ScupButton btState = new ScupButton(this);
		// Set up icon
		btState.setIcon(R.drawable.states);
		// View of button
		btState.setAlignment(ScupButton.ALIGN_VERTICAL_CENTER);
		btState.setPadding(5, 0, 5, 0);
		// Text parameter
		btState.setText("State");
		btState.setTextSize(5);
		// Set up a click listener
		btState.setClickListener(new ScupButton.ClickListener() {
			@Override
			public void onClick(ScupButton arg0) {
				i= (i+1)%2;
				btState.setBackgroundColor(color[i]);
				update();
			}
		});
		// Show this button
		btState.show();

		// Music button
		ScupButton btMusic = new ScupButton(this);

		btMusic.setIcon(R.drawable.music);
		btMusic.setAlignment(ScupButton.ALIGN_VERTICAL_CENTER);
		btMusic.setPadding(5, 0, 5, 0);

		btMusic.setText("Music");
		btMusic.setTextSize(5);

		btMusic.setClickListener(new ScupButton.ClickListener() {
			@Override
			public void onClick(ScupButton arg0) {
				showToast("Music button is clicked", Toast.LENGTH_LONG);
			}
		});
		btMusic.show();
	}
}
```

4.2.1. Creating the Dialog Subclass

To use any widget, create a dialog:

1. Create a class extending from the ScupDialog base class (ScupButtonSampleApp in the example).
2. Implement the constructor for the class.

```
public class CupButtonSampleApp extends ScupDialog {

	private int mPageIndex = 0;
	private final ArrayList<ScupButton> mButtons = new ArrayList<ScupButton>();

	public CupButtonSampleApp(Context context) {
		super(context);
	}
```

3. Declare and initialize widgets in the `onCreate()` method.

```
@Override
	protected void onCreate() {
		super.onCreate();
		// Update the view in each status
		update(mPageIndex);
		// Animation of show
		setAnimationType(ANIMATION_ALPHA);
	}
```

In the example, the `update()` method updates the buttons view on the Wingtip screen. You update the mPageIndex parameter to change the view.

4.2.2. Adding the Button Widget

To add a new instance of ScupButton:

1. Call the ScupButton class constructor and pass the dialog object (this class) into the parameter for the method.
2. Set ScupButton attributes with, for example, the `setIcon()`, `setAlignment()`, `setPadding()`, `setTextSize()`, and `setText()` methods.
   Note: By default, the button icon is to the left of the button text.

The CUP package supports the following ScupButton alignments:

Table 1: ScupButton alignments

Type	Description
ALIGN_TOP	Aligns contents with the top edge of the widget area.
ALIGN_BOTTOM	Aligns contents with the bottom edge of the widget area.
ALIGN_VERTICAL_CENTER	Aligns contents with the vertical center of the widget area.
ALIGN_LEFT	Aligns contents with the left edge of the widget area.
ALIGN_RIGHT	Aligns contents with the right edge of the widget area.

Type	Description
ALIGN_HORIZONTAL_CENTER	Aligns contents with the horizontal center of the widget area.
ALIGN_CENTER	Aligns contents with the center of the widget area.

The CUP package supports the following ScupButton sizes:

Table 2: ScupButton sizes

Type	Value	Description
FILL_DIALOG	0	The widget width or height must be the same as the dialog's width or height.
WRAP_CONTENT	1	The widget width or height must fit its internal content.

```
btState.setIcon(R.drawable.states);
              // View of button
              btState.setAlignment(ScupButton.ALIGN_VERTICAL_CENTER);
              btState.setPadding(5, 0, 5, 0);
              // Text parameter
              btState.setText("State");
              btState.setTextSize(5);
              // Set up a click listener
              btState.setClickListener(new ScupButton.ClickListener() {
                    @Override
                    public void onClick(ScupButton arg0) {
                          i= (i+1)%2;
                          btState.setBackgroundColor(color[i]);
                          update();
                    }
              });
              // Show this button
              btState.show();
```

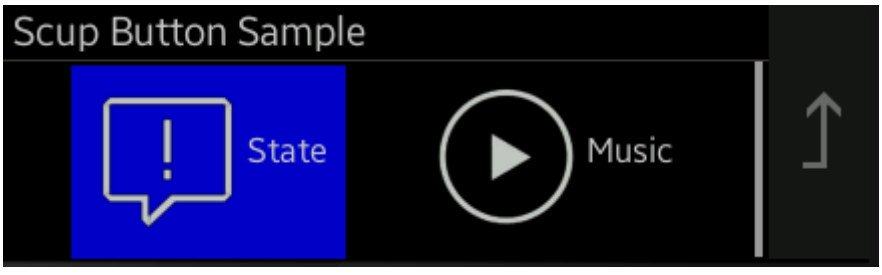

Figure 9: Button example

```
              // Music button
              ScupButton btMusic = new ScupButton(this);

              btMusic.setIcon(R.drawable.music);
              btMusic.setAlignment(ScupButton.ALIGN_VERTICAL_CENTER);
              btMusic.setPadding(5, 0, 5, 0);
```

```
btMusic.setText("Music");
btMusic.setTextSize(5);

btMusic.setClickListener(new ScupButton.ClickListener() {
	@Override
	public void onClick(ScupButton arg0) {
		showToast("Music button is clicked", Toast.LENGTH_LONG);
	}
});
btMusic.show();
```

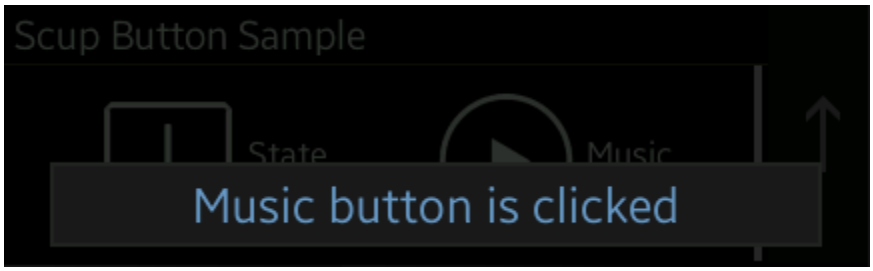

Figure 10: Clicked button example

ScupButton.ClickListener is called when the user clicks the button. To add an event to the button, call the setClickListener() method, pass an instance ScupButton.ClickListener to the parameter, and implement the `onClick()` method for the event.

```
btMusic.setClickListener(new ScupButton.ClickListener() {
	@Override
	public void onClick(ScupButton arg0) {
		showToast("Music button is clicked", Toast.LENGTH_LONG);
	}
});
```

4.3. Using Graphs

The ScupGraph class represents a graph widget, which is a kind of extended widget. If the device cannot support this widget, it can display a base widget instead. To use the graph:

1. Create a subclass of the ScupDialog class.
2. Add the ScupGraph widget to the application.
   You can create a graph with a maximum of 10 columns and 2 graph styles.

The following figure and code example illustrate the ScupGraphDialog sample application that creates a ScupGraph instance in the dialog with some graph-specific attributes.

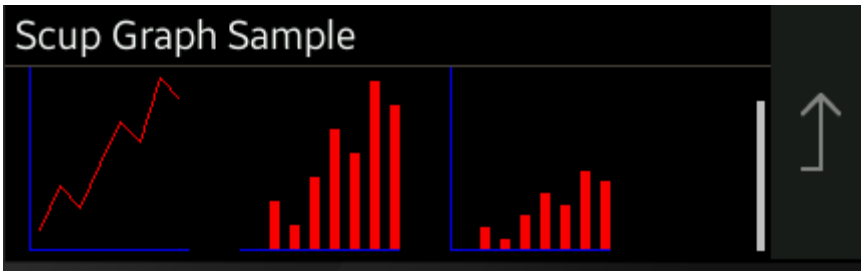

Figure 11: ScupGraph

```
public class ScupGraphDialog extends ScupDialog{

	public int data[] = {10,30,20,40,60,50,80,70,90};
	public ScupGraphDialog(Context context) {
		super(context);
	}
	@Override
	protected void onCreate() {
		super.onCreate();
		update(0);
		setAnimationType(ANIMATION_ALPHA);
	}
	void update(int index) {
		this.setTitle("Scup Graph Sample ");
		// Create first graph
		ScupGraph mGraph = new ScupGraph(this);
		// Add data to graph
		for(int i=0;i<8;i++)
		{
			mGraph.addData(i,data[i], Color.RED, null, null);
		}
		// Set up the parameter
		// Color of Axis and
		mGraph.setAxisColor(Color.BLUE, Color.GREEN);
		// Style of Graph
		mGraph.setStyle(ScupGraph.STYLE_LINE);
		// Set Y axis - maximum , minimum value and step
		mGraph.setYAxis(0, 100, 10);
		mGraph.show();

		ScupGraph mGraph2 = new ScupGraph(this);
		for(int i=0;i<8;i++)
		{
			mGraph2.addData(i,data[i], Color.RED,null, null);
		}
		mGraph2.setAxisColor(Color.BLUE, Color.GREEN);
		mGraph2.setStyle(ScupGraph.STYLE_BAR);
		mGraph2.setYAxis(10, 100, 10);
		// Hide Y axis in view
		mGraph2.hideYAxis();
		// Set image at the point value
		mGraph2.setPointImage(R.drawable.red);
		mGraph2.show();

		ScupGraph mGraph3 = new ScupGraph(this);
		for(int i=0;i<8;i++)
		{
			mGraph3.addData(i,data[i], Color.RED,null, null);
```

```
            }
            mGraph3.setAxisColor(Color.BLUE, Color.GREEN);
            mGraph3.setStyle(ScupGraph.STYLE_BAR);
            // Set different at Y axis
            mGraph3.setYAxis(10, 200, 20);
            mGraph3.showYAxis();
            mGraph3.show();
      }
}
```

Create a new ScupGraph instance and add the value of each column to the graph with the `addData(int id, int value, int color, String xAxisText, String valueText)` method, whose parameters are:

- **`id`**: The ID of the data.
- **`value`**: The value of the data.
- **`color`**: The bar line color.
- **`xAxisText`**: The text under the X axis.
- **`valueText`**: The text near the data point.

```
      // Create first graph
       ScupGraph mGraph = new ScupGraph(this);
       // Add data to graph
       for(int i=0;i<8;i++)
       {
             mGraph.addData(i,data[i], Color.RED, null, null);
       }
```

Set the graph attributes:

- **setAxisColor**(): Sets the baseline color and text color.
- **setStyle**(): Sets the style of graph.
  Two graph styles are available:
  - STYLE_BAR: The graph points are shown as bar columns.
  - STYLE_LINE (default): The graph points are connected with a straight line.

  At most 9 points are shown in the graph.
- **setYAxis**(): Sets the minimum and maximum values for the Y axis, and the horizontal line drawn for each step value.
- **setPointImage**(): Sets the image and color to be displayed in the graph.

```
       // Set up the parameter
       // Color of Axis and
       mGraph.setAxisColor(Color.BLUE, Color.GREEN);
       // Style of Graph
       mGraph.setStyle(ScupGraph.STYLE_LINE);
       // Set Y axis - maximum , minimum value and step
```

```
        mGraph.setYAxis(0, 100, 10);
       // Set Y axis - maximum , minimum value and step
        mGraph.setYAxis(0, 100, 10);
         // Set point image in graph
       mGraph2.setPointImage(R.drawable.red);
```

Note: If the `showYAxis()` method is not set up for the graph, the Y axis is shown by default. You can hide the Y axis with the `hideYAxis()` method.

4.4. Using Labels

The ScupLabel class represents a label widget. To use the label:

1. Create a subclass of ScupDialog class.
2. Add the ScupLabel widget to the dialog with parameters.
   The ScupLabel has the same methods as the TextView view in Android. The TextView methods include `setWidth()`, `setHeight()`, `setText()`, `setTextSize()`, and `setTextColor()`.
3. Add the label dialog to the main application.

The following figure and code example illustrate the ScupLabelDialog sample application that creates a ScupLabel instance in the dialog with some label-specific attributes.

Figure 12: ScupLabel

```
package com.example.scupsamplev1;

import android.content.Context;
import android.graphics.Color;

import com.samsung.android.sdk.cup.ScupDialog;
import com.samsung.android.sdk.cup.ScupLabel;

public class ScupLabelDialog extends ScupDialog {

	private ScupLabel label1;
	private ScupLabel label2;

	public ScupLabelDialog(Context context) {
		super(context);
	}

	@Override
```

```
protected void onCreate() {
	super.onCreate();

	// Make an instance of SCupLabel object
label1 = createLabel(this, "Label 1", ScupLabel.ALIGN_CENTER,
		0xFFFFFFFF, R.drawable.sample);

if (label1 != null) {
	// Set type and color of border for label control
	label1.setBorderType(ScupLabel.BORDER_TYPE_SQUARE);
	label1.setBorderColor(Color.WHITE);
	// Set icon position for label
	label1.setIconPosition(ScupLabel.POSITION_ICON_ABOVE);
	// Show the label control
	label1.show();

	// Add a listener for label control
	label1.setTapListener(new ScupLabel.TapListener() {

	@Override
	public void onSingleTap(ScupLabel arg0, float arg1, float arg2) {
	// TODO Auto-generated method stub
	toast("Label 1 Tapped !, X1 = " + arg1 + ", Y1 = " + arg2);
			}
		});
	}

	// Create a second label, the same to it
	label2 = createLabel(this, "Label 2", ScupLabel.ALIGN_HORIZONTAL_CENTER
	| ScupLabel.ALIGN_BOTTOM, 0xFFFFFFFF, R.drawable.button_ok);

if (label2 != null) {
	label2.setIconPosition(ScupLabel.POSITION_ICON_LEFT);
	label2.setBorderType(ScupLabel.BORDER_TYPE_INTAGLIO);
	label2.setBorderColor(Color.WHITE);
	label2.show();

	label2.setTapListener(new ScupLabel.TapListener() {

	@Override
	public void onSingleTap(ScupLabel arg0, float arg1, float arg2) {
	// TODO Auto-generated method stub
	toast("Label 2 Tapped !,X2 = " + arg1 + ", Y2 = " + arg2);
		}
	});

	}

	// Set background for dialog control
	this.setBackgroundColor(0x000000);
	// Set the gap of controls in the dialog
	this.setWidgetGap(5);
	// Set title of the dialog
	this.setTitle("Scup Label Sample");
	// Show the dialog
	}

/**
 * method for creating a label widget
```

```
         *
         * @param scupDialog
         * @param txt
         * @param align
         * @param color
         * @param iconId
         * @return
         */
        private ScupLabel createLabel(ScupDialog scupDialog, String txt, int align,
                    int color, int iconId) {

              ScupLabel scupLabel = new ScupLabel(scupDialog);
              scupLabel.setWidth(ScupLabel.WRAP_CONTENT);
              scupLabel.setHeight(ScupLabel.WRAP_CONTENT);

              scupLabel.setIcon(iconId);
              scupLabel.setTextSize(6);
              scupLabel.setTextColor(color);
              scupLabel.setText(txt);
              scupLabel.setAlignment(align);
              scupLabel.setSingleLineModeEnabled(true);

              return scupLabel;
        }

        /**
         * show a toast
         *
         * @param msg
         *            the text to show
         */
        private void toast(String msg) {
              this.showToast(msg, ScupDialog.TOAST_DURATION_SHORT);
        }

        @Override
        protected void onDestroy() {
              // TODO super extend
              // Destroy label control after using finish for avoid memory leak
              if (label1 != null) {
                    label1.destroy();
              }
              if (label2 != null) {
                    label2.destroy();
              }

              super.onDestroy();
        }
}
```

4.4.1. Creating the Dialog Subclass

To use any widget, create a dialog:

1. Create a class extending from the ScupDialog base class (ScupLabelDialog in the example).
2. Implement the constructor for the class.

```
public class  SCupLabelDialog extends ScupDialog {

	private final Context mContext;
	 private ScupLabel label1;
	private ScupLabel label2;


	public  ScupLabelDialog (Context context) {
		super(context);
		mContext = context;
	}
```

3. Inside the constructor of the class, call the constructor of the base class.

4. Declare and initialize widgets in the `onCreate()` method.

```
@Override
	protected void onCreate() {
		super.onCreate();

		label1 = createLabel(this, "Label 1",
		ScupLabel.ALIGN_CENTER, 0xFFFFFFFF, R.drawable.sample);
```

5. Set, for example, the title and the gap between the widgets in the dialog.

```
	// Set background for dialog widget
	 this.setTitle("Scup Label Sample");
	this.setWidgetGap(5);
```

4.4.2. Adding the Label Widget

To add new instance of ScupLabel:

1. Create an ScupLabel class instance. Inside the ScupLabel widget constructor, pass the dialog object, in this case scupDialog.

```
	 ScupLabel scupLabel = new ScupLabel(scupDialog);
```

2. Set ScupLabel attributes.

```
		 // Set size for label: width and height
		scupLabel.setWidth(ScupLabel.WRAP_CONTENT);
		scupLabel.setHeight(ScupLabel.WRAP_CONTENT);
		 // Set icon for label
		scupLabel.setIcon(iconId);
		 // Set text properties for label
		scupLabel.setTextSize(6);
		scupLabel.setTextColor(color);
		scupLabel.setText(txt);
		scupLabel.setAlignment(align);
		scupLabel.setSingleLineModeEnabled(true);
```

3. Set the label width and height.

```
scupLabel.setWidth(ScupLabel.WRAP_CONTENT);
scupLabel.setHeight(ScupLabel.WRAP_CONTENT);
```

The SCupLabel class supports the following width and height values:

Table 3: SCupLabel width and height attributes

Type	Value	Description
WRAP_CONTENT	-2	The widget width or height must fit its internal content.
FILL_DIALOG	-1	The widget width or height must be the same as the dialog width and height.

Note: If the `setWidth()` and `setHeight()` methods are not available, both are set to WRAP_CONTENT by default.

4. Set the label icon position.

```
label2.setIconPosition(ScupLabel.POSITION_ICON_ABOVE);
```

The SCupLabel class supports the following icon positions:

Table 4: SCupLabel icon positions

Type	Value	Description
POSITION_ICON_LEFT	1	The icon is to the left of text.
POSITION_ICON_RIGHT	2	The icon is to the right of text.
POSITION_ICON_ABOVE	3	The icon is above text.
POSITION_ICON_BELOW	4	The icon is below text.

5. Set the label border type.

```
label1.setBorderType(ScupLabel.BORDER_TYPE_SQUARE);
```

The SCupLabel class supports the following label border types:

Table 5: Label border types

Type	Value	Description
BORDER_TYPE_SHADOW	3	The shadow border type.
BORDER_TYPE_SQUARE	1	The square border type.
BORDER_TYPE_NONE	0	No border.
BORDER_TYPE_INTAGLIO	2	The intaglio border type.
BORDER_TYPE_UNDERLINE	4	The underline border type.

6. Check whether the label is created successfully.

```
if (label1 != null) {
              //.....................................
       label1.show();
```

7. Add a tap event to the label widget with the ScupLabel.TapListener interface.
    - Call `setTapListener()`.

- Create a new instance of the ScupLabel.TapListener listener and register it by passing it into the setTapListener() method.
- Implement the onSingleTap() method of the listener inside the method, in this case showing a toast by calling the toast() method.

```
	// Add a listener for label widget
	label1.setTapListener(new ScupLabel.TapListener() {

		@Override
		public void onSingleTap(ScupLabel label, float x, float y) {
			// TODO Auto-generated method stub
			toast("Label 1 Tapped !, X1 = " + x + ", Y1 = " + y);
		}
	});


private void toast(String msg) {
		this.showToast(msg, ScupDialog.TOAST_DURATION_SHORT);
	}
```

8. After using the label widget `finish()` method, call `destroy()` for the label in the overridden `onDestroy()` method of the ScupDialog base class.

```
	@Override
	protected void onDestroy() {

		if (label1 != null) {
			label1.destroy();
		}
		if (label2 != null) {
			label2.destroy();
		}
		super.onDestroy();
	}
```

4.4.3. Adding the Label Dialog to the Application

To use the CupLabelDialog class in the MainActivity, create an instance of ScupLabelDialog (mCupLabelDialog), and pass ApplicationContext in the constructor of the class, if it is null. If it is not null, call the `finish()` method and assign it to null.

```
// mCupLabelDialog is a ScupLabelDialog  object
	if (mCupLabelDialog == null) {
	mCupLabelDialog = new ScupLabelDialog(getApplicationContext());
	} else {
	mCupLabelDialog.finish();
```

For more information, see the CupLabelDialog.java code file.

4.5. Using Spinners

The ScupSpinner is a subclass of ScupWidgetBase, and represents a spinner widget. It helps the user to select an item with the plus and minus buttons from a list of items with a simple UI, and supports callbacks when the user selects an item or changes a selected item. To use the spinner:

1. Create a subclass of the ScupDialog class.
2. Add the ScupSpinner widget to the application.

The following figure and code example illustrate the ScupSpinnerDisplay sample application that creates a ScupSpinner instance in the dialog with some spinner-specific attributes.

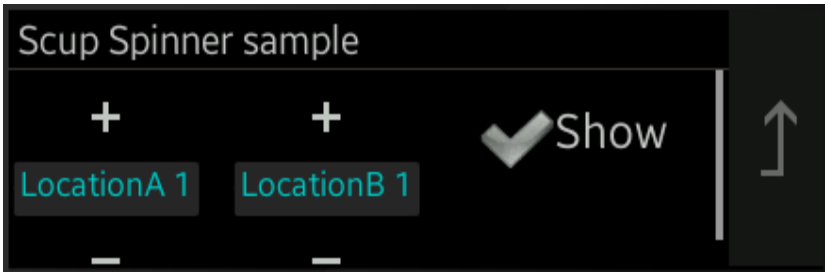

Figure 13: ScupSpinner

```
public class ScupSpinnerDisplay extends ScupDialog {

	private ScupSpinner spinner;
	private ScupSpinner scupSpinner;
	private ScupButton scupButton;
	private int x1 = 1;
	private int x2 = 1;
	private final ScupDialog dialog;

	public ScupSpinnerDisplay(Context context) {
		super(context);
		dialog = this;
	}

	@Override
	protected void onCreate() {
		super.onCreate();
		// create two scupspinners instance using a ScupDialog
		spinner = new ScupSpinner(this, ScupSpinner.STYLE_VERTICAL);

		// Set width and height of spinner
		// Width and height params can be Wrap_Content or FILL_DIALOG
		spinner.setWidth(ScupSpinner.WRAP_CONTENT);
		spinner.setHeight(ScupSpinner.WRAP_CONTENT);

		// Select text color and size
		spinner.setTextColor(Color.CYAN);
		spinner.setTextSize(5);

		// Set padding of spinner
		spinner.setPadding(1, 1, 1, 1);
```

```
// Add each item of spinner for user to select
// Need : item id, item text, and bitmap icon if user need image
spinner.addItem(1, "LocationA 1", null);
spinner.addItem(2, "LocationA 2", null);
spinner.addItem(3, "LocationA 3", null);

spinner.setChangeListener(new ScupSpinner.ChangeListener() {

	@Override
	public void onChanged(ScupSpinner arg0, int arg1) {
		x1 = arg1;
		dialog.showToast("SpinnerA change to LocationA " + arg1,
				ScupDialog.TOAST_DURATION_SHORT);
	}
});
// Set onClick listener
// Whenever user click on the item of spinner
spinner.setClickListener(new ClickListener() {

	@Override
	public void onClick(ScupSpinner spinner, int itemId) {
		dialog.showToast("The current item: LocationA " + itemId,
				ScupDialog.TOAST_DURATION_SHORT);
	}
});

// The same to this ScupSpinner
scupSpinner = new ScupSpinner(this, ScupSpinner.STYLE_VERTICAL);
scupSpinner.addItem(1, "LocationB 1", null);
scupSpinner.addItem(2, "LocationB 2", null);
scupSpinner.addItem(3, "LocationB 3", null);
scupSpinner.setWidth(ScupSpinner.WRAP_CONTENT);
scupSpinner.setHeight(ScupSpinner.WRAP_CONTENT);
scupSpinner.setTextColor(Color.CYAN);
scupSpinner.setTextSize(5);
scupSpinner.setPadding(1, 1, 1, 1);
scupSpinner.setChangeListener(new ScupSpinner.ChangeListener() {

	@Override
	public void onChanged(ScupSpinner arg0, int arg1) {
		x2 = arg1;
		dialog.showToast("SpinnerB change to LocationB " + arg1,
				ScupDialog.TOAST_DURATION_SHORT);
	}
});

// Spinner.show() must be call to show the spinner with the parent
// dialog;
spinner.show();
scupSpinner.show();

// Create an instance of scupbutton and set some attributes and event
// for it
scupButton = new ScupButton(this);
scupButton.setWidth(ScupButton.WRAP_CONTENT);
scupButton.setHeight(ScupButton.WRAP_CONTENT);
scupButton.setText("Show");
scupButton.setBackgroundColor(Color.GREEN);
```

```
            scupButton.setClickListener(new ScupButton.ClickListener() {

                  @Override
                  public void onClick(ScupButton arg0) {
                        dialog.showToast("From LocationA " + x1 + ", To LocationB "
                                    + x2, ScupDialog.TOAST_DURATION_SHORT);
                  }
            });
            // Show the scupbutton on WT screen
            scupButton.show();

            // set some attributes for dialog container
            this.setWidgetGap(5);
            this.setTitle("Scup Spinner sample");

      }

      @Override
      protected void onDestroy() {


            // Destroy all widgets to avoid memory leak
            if (scupSpinner != null) {
                  scupSpinner.removeItemAll();
                  scupSpinner.destroy();
            }

            if (spinner != null) {
                  spinner.removeItemAll();
                  spinner.destroy();
            }
            super.onDestroy();
      }
}
```

4.5.1. Creating the Dialog Subclass

To use any widget, create a dialog:

1. Create a class extending from the ScupDialog base class (ScupSpinnerDisplay in the example).
2. Implement the constructor for the class.

```
public class ScupSpinnerDisplay extends ScupDialog {

       private ScupSpinner spinner;
      private ScupSpinner scupSpinner;
      private ScupButton scupButton;
      private int x1 = 1;
      private int x2 = 1;
     private final ScupDialog dialog;

      public ScupSpinnerDisplay(Context context) {
            super(context);
            dialog = this;
      }
```

3. Override the `onCreate()` method from the base class.

```
@Override
protected void onCreate() {
  super.onCreate();
 //.....................................
```

4.5.2. Adding the Spinner Widget

To add a new instance of ScupSpinner:

1. Call the ScupSpinner class constructor and pass the dialog object (ScupSpinnerDisplay in this example) into the parameter for the method.
2. Set ScupSpinner attributes.

   The ScupSpinner class supports the following spinner styles, which are set when creating the ScupSpinner instance.

Table 6: ScupSpinner styles

Type	Value	Description
STYLE_VERTICAL	1	The vertical spinner style.
STYLE_HORIZONTAL	2	The horizontal spinner style. This is the default style.

```
            // create a scup spinner instance using a Scup Dialog
            ScupSpinner spinner = new ScupSpinner(this, ScupSpinner.STYLE_VERTICAL);

            // set width and height of spinner
            // width and height params can be Wrap_Content or Match_Parent
            spinner.setWidth(ScupSpinner.WRAP_CONTENT);
            spinner.setHeight(ScupSpinner.WRAP_CONTENT);

            // select text color and size
            spinner.setTextColor(Color.CYAN);
            spinner.setTextSize(10);

            // set padding of spinner
            spinner.setPadding(2, 2, 2, 2);
            //.........................
            .........
            spinner.show();
```

3. To set the spinner size, call the `setWidth()` and `setHeight()` methods.

```
            // set width and height of spinner
            // width and height params can be Wrap_Content or Match_Parent
            spinner.setWidth(ScupSpinner.WRAP_CONTENT);
            spinner.setHeight(ScupSpinner.WRAP_CONTENT);
```

The ScupSpinner class supports the following spinner sizes.

Table 7: ScupSpinner sizes

Type	Value	Description
WRAP_CONTENT	-2	The widget width or height must fit its internal content.
FILL_DIALOG	-1	The widget width or height must be the same as the dialog width or height.

4. Give the ScupSpinner property values to display the spinner as expected.
    - `setTextSize()`: Size of the text.
    - `setTextColor()`: Color of the text.
    - `setPadding()`: Padding of the widget.

```
// select text color and size
spinner.setTextColor(Color.CYAN);
spinner.setTextSize(5);
spinner.setCirculationEnabled(false);
// set padding of spinner
spinner.setPadding(1, 1, 1, 1);
```

5. Add items to the spinner item list with the `addItem()` method of ScupSpinner.

```
// add each item of spinner for user to select
// need : item id, item text, and bitmap icon if user need image
spinner.addItem(1, "LocationA 1", null);
spinner.addItem(2, "LocationA 2", null);
spinner.addItem(3, "LocationA 3", null);
```

6. Remove an item using its ID, or remove all items.

```
spinner.removeItem(id);
 spinner.removeItemAll();
```

7. Set the listeners for the spinner widget:
    - ClickListener: Called when the user clicks on the spinner list item.
    - ChangedListener: Called when the user selects another item by clicking the widget's up or down arrows.

    Register a click event by calling `Spinner.setClickListener()`, passing the parameter to this method by creating a new instance of ScupSpinner.ClickListener. You must override with the `onClick()` method.

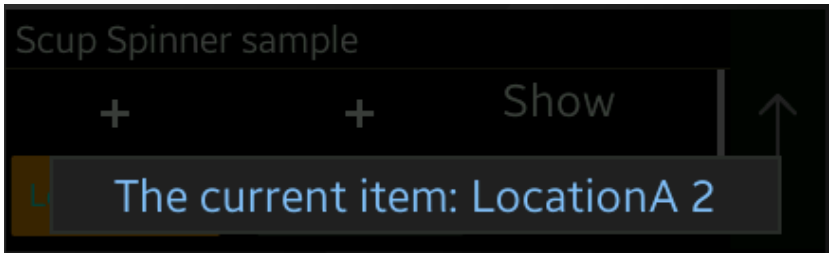

Figure 14: ScupSpinner - ClickListener

To handle events when an item is changed, use `SCupSpinner.setChangeListener()`, then implement the `onChanged()` method. The following sample code shows a toast.

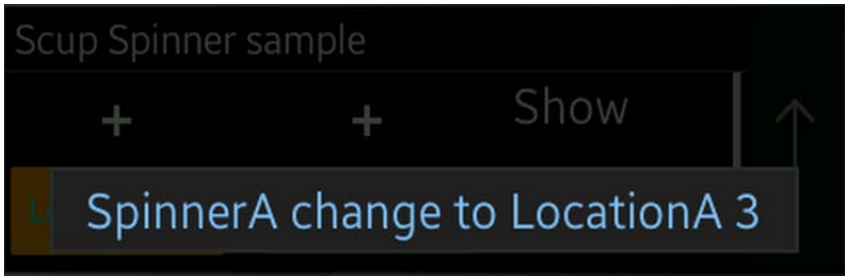

Figure 15: ScupSpinner – ChangeListener

```
// Handle listener ScupSpinner
spinner.setChangeListener(new ScupSpinner.ChangeListener() {

    @Override
    public void onChanged(ScupSpinner arg0, int arg1) {
            x1 = arg1;
            dialog.showToast("SpinnerA change to LocationA " + arg1,
                                    ScupDialog.TOAST_DURATION_SHORT);
      }
    });
    // Set onClick listener
    // Whenever user click on the item of spinner
    spinner.setClickListener(new ClickListener() {

      @Override
      public void onClick(ScupSpinner spinner, int itemId) {
            dialog.showToast("The current item: LocationA " + itemId,
                                    ScupDialog.TOAST_DURATION_SHORT);
      }
  });
```

For more information, see the ScupSpinnerDisplay.java code file.

4.6. Using Lists

The ScupListBox class represents a list widget, which is a kind of base widget. To use the list:

1. Create a subclass of the ScupDialog class.
2. Add the ScupListBox widget to the application.

The following figure and code example illustrate the ScupListBoxDialog sample application that creates a ScupListBox instance in the dialog with some list-specific attributes.

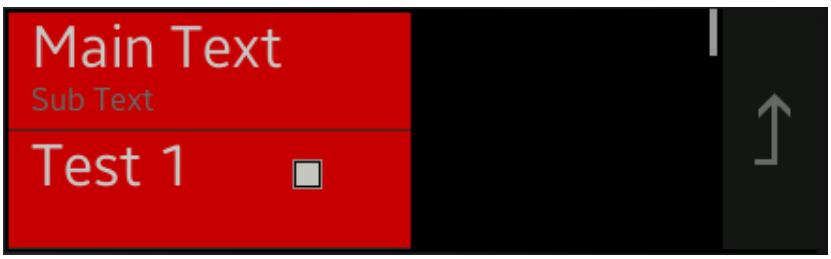

Figure 16: ScupListBox

```
public class ScupListBoxDialog extends ScupDialog {
```

```
private ScupListBox mListBox;

public ScupListBoxDialog(Context context) {
	super(context);
}

@Override
protected void onCreate() {
	super.onCreate();

	// Create a ScupListBox
	mListBox = new ScupListBox(this);

	// Add items to ScupListBox
	mListBox.addItem(0, "Test 0");
	mListBox.addItem(1, "Test 1");
	mListBox.addItem(2, "Test 2");
	mListBox.addItem(3, "Test 3");
	mListBox.addItem(4, "Test 4");
	mListBox.addItem(5, "Test 5");

	// Remove all Items
	mListBox.removeItemAll();

	mListBox.addItem(0, "Test 0");
	mListBox.addItem(1, "Test 1");
	mListBox.addItem(2, "Test 2");
	mListBox.addItem(3, "Test 3");
	mListBox.addItem(4, "Test 4");
	mListBox.addItem(5, "Test 5");
	mListBox.addItem(6, "Test 6");
	mListBox.addItem(7, "Test 7");
	mListBox.addItem(8, "Test 8");
	mListBox.addItem(9, "Test 9");

	// Insert an item to ScupListBox
	mListBox.insertItem(7, 10, "Test 10");

	// Remove an Item
	mListBox.removeItem(4);

	// Set enable/disable an Item
	mListBox.setItemEnabled(6, false);

	// Set main text, sub text of Item
	mListBox.setItemMainText(0, "Main Text");
	mListBox.setItemSubText(0, "Sub Text");

	mListBox.setItemMainTextSize(9);
	mListBox.setItemSubTextSize(5);

	// Set type of Item
	mListBox.setItemButtonType(0, ScupListBox.ITEM_BUTTON_NONE);
	mListBox.setItemButtonType(1, ScupListBox.ITEM_BUTTON_CHECK);
	mListBox.setItemButtonType(2, ScupListBox.ITEM_BUTTON_MORE);
	mListBox.setItemButtonType(3, ScupListBox.ITEM_BUTTON_RADIO);

	mListBox.setItemButtonType(5, ScupListBox.ITEM_BUTTON_USER);
```

```
            mListBox.setItemButtonType(7, ScupListBox.ITEM_BUTTON_RADIO);
            mListBox.setItemButtonType(8, ScupListBox.ITEM_BUTTON_RADIO);
            mListBox.setItemButtonType(9, ScupListBox.ITEM_BUTTON_RADIO);
            mListBox.setItemButtonType(10, ScupListBox.ITEM_BUTTON_CHECK);

            // Set group of Items
            mListBox.setItemButtonGroup(7, 100);
            mListBox.setItemButtonGroup(8, 100);
            mListBox.setItemButtonGroup(9, 100);

            // Set select/deselect an item
            mListBox.selectItemButton(0);
            mListBox.deselectItemButton(3);

            mListBox.setItemBackgroundColor(Color.RED);

            // Set Text for Item button
            mListBox.setItemButtonText(5, "Item 5");

            // Set Width/height of ListBox
            mListBox.setWidth(ScupListBox.WRAP_CONTENT);
            mListBox.setHeight(ScupListBox.WRAP_CONTENT);

            // Set Listener of ListBox
            mListBox.setItemClickListener(new ItemClickListener() {

                  @Override
                  public void onClick(ScupListBox list, int id, int groupid,
                              boolean buttonState) {
                        showToast("Item " + id + " is clicked !!!",
                                    ScupDialog.TOAST_DURATION_SHORT);
                  }
            });

            // Show ListBox
            mListBox.show();

      }

      @Override
      public void finish() {
            super.finish();
            if (mListBox != null) {
                  // Destroy a ListBox
                  mListBox.destroy();
                  mListBox = null;
            }
      }

      @Override
      protected void onDestroy() {
            super.onDestroy();
            finish();
      }
}
```

4.6.1. Creating the Dialog Subclass

To use any widget, create a dialog:

1. Create a class extend from the ScupDialog base class (ListBoxDialog in the example).
2. Implement the constructor for the class.

```
public class ListBoxDialog extends ScupDialog {

	private ScupListBox mListBox;

	public ListBoxDialog(Context context) {
		super(context);
	}
```

3. Override the `onCreate()` method from the base class.

```
	@Override
	protected void onCreate() {
		super.onCreate();

		// Create a ScupListBox
		mListBox = new ScupListBox(this);
```

4.6.2. Adding the List Widget

To add a new instance of ScupListBox:

1. Call the ScupListBox class constructor and pass the dialog object (this class) into the parameter for the method.
2. Set ScupListBox attributes with, for example, the `addItem()`, `removeItem()`, `setItemMainText()`, `setItemSubText()`, `setItemButtonType()`, or `setItemBackgroundColor()` methods.

```
		// Set type of Item
		mListBox.setItemButtonType(0, ScupListBox.ITEM_BUTTON_NONE);
		mListBox.setItemButtonType(1, ScupListBox.ITEM_BUTTON_CHECK);
		mListBox.setItemButtonType(2, ScupListBox.ITEM_BUTTON_MORE);
		mListBox.setItemButtonType(3, ScupListBox.ITEM_BUTTON_RADIO);

		mListBox.setItemButtonType(5, ScupListBox.ITEM_BUTTON_USER);

		mListBox.setItemButtonType(7, ScupListBox.ITEM_BUTTON_RADIO);
		mListBox.setItemButtonType(8, ScupListBox.ITEM_BUTTON_RADIO);
		mListBox.setItemButtonType(9, ScupListBox.ITEM_BUTTON_RADIO);
		mListBox.setItemButtonType(10, ScupListBox.ITEM_BUTTON_CHECK);
```

The SCupListBox class supports the following list item types:

Table 8: ScupListBox types

Type	Value	Description
ITEM_BUTTON_NONE	0	No button image to the right of the list item.
ITEM_BUTTON_CHECK	1	A default check button image to the left of the list item.
ITEM_BUTTON_RADIO	2	A default radio button image to the right of the list item.
ITEM_BUTTON_MORE	3	A default more button image to the right of the list item.
ITEM_BUTTON_SLIDE	4	Not supported yet.
ITEM_BUTTON_TEXT	5	A text area to the right of the list item.
ITEM_BUTTON_USER	6	A user-defined image to the right of the list item.

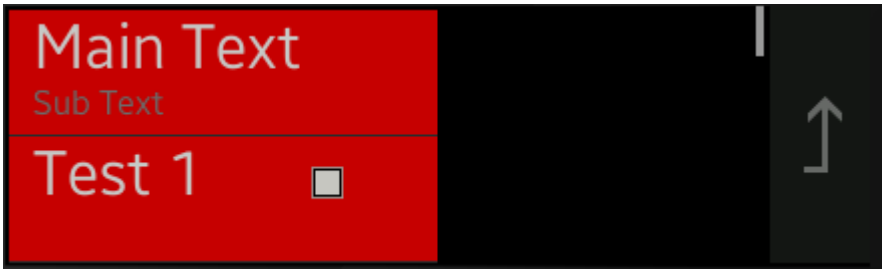

Figure 17: **ScupListBox type 1**

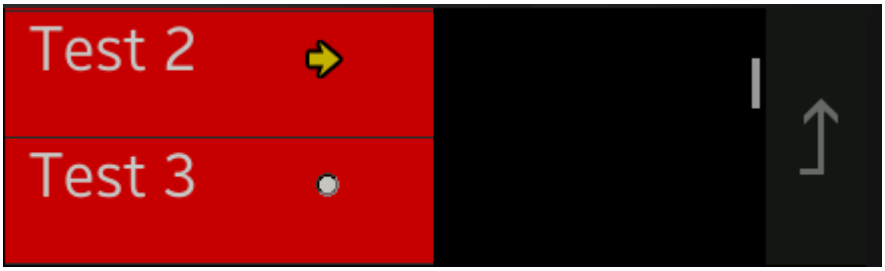

Figure 18: ScupListBox type 2

3. Set width and height of ListBox.

```
// Set Width/height of ListBox
mListBox.setWidth(ScupListBox.FILL_DIALOG);
mListBox.setHeight(ScupListBox.WRAP_CONTENT);
```

The SCupListBox class supports the following list sizes:

Table 9: ScupListBox sizes

Type	Value	Description
FILL_DIALOG	0	The widget width or height is the same as the dialog width or height.
WRAP_CONTENT	1	The widget width or height fits its internal content.

Note: If the width and height are positive values between 1 and 100, they refer to a percentage of the screen width or screen height. For example, if you use `setWidth(50)`, the width of the widget is half of the screen width.

4. Set a listener for click events when the user clicks a list item.
   To add an event to ScupListBox, call the `setItemClickListener()` method and pass a ScupListBox.ItemClickListener instance to the parameter. In ScupListBox.ItemClickListener, implement the `onClick()` method to handle the event.

```
// Set Listener of ListBox
mListBox.setItemClickListener(new ItemClickListener() {

      @Override
      public void onClick(ScupListBox list, int id, int groupid,
                  boolean buttonState) {
      showToast("Item " + id + " is clicked !!!",
                        ScupDialog.TOAST_DURATION_SHORT);
      }
});
```

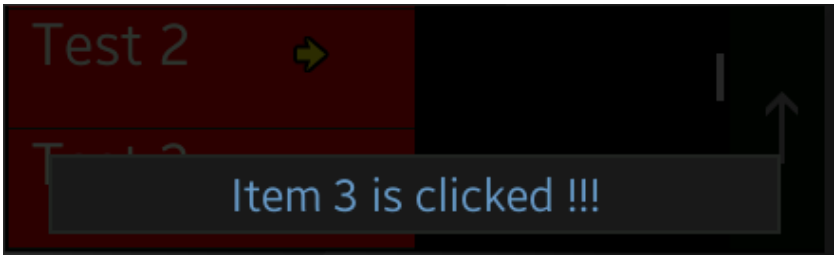

Figure 19**: ScupListBox - ClickListener**

4.7. Using Progress Bars

The ScupProgressBar class represents a progress bar widget, which is a kind of extended widget. If a device cannot support this widget, it can display a base widget instead. To use the progress bar:

1. Create a subclass of the ScupDialog class.
2. Add the ScupProgressBar widget to the application.

The following figure and code example illustrate the ScupProgressBarDialog sample application that creates a ScupProgressBar instance in the dialog with some progress bar-specific attributes.

Figure 20: **ScupProgressBar**

```
public class SCupProgressBarDialog extends ScupDialog {

	private ScupProgressBar progressBar;
	int progressStatus = 0;

	public SCupProgressBarDialog(Context context) {
		super(context);
	}

	@Override
	protected void onCreate() {
		super.onCreate();

		setTitle("Scup ProgresBar Sample");
		setBackgroundColor(Color.LTGRAY);

		setWidgetAlignment(WIDGET_ALIGN_HORIZONTAL_CENTER
				| WIDGET_ALIGN_VERTICAL_CENTER);

		// Create a progressbar with style
		progressBar = new ScupProgressBar(this, ScupProgressBar.STYLE_IMAGE);

		// Set width/ height of progressbar
		progressBar.setHeight(ScupProgressBar.WRAP_CONTENT);
		progressBar.setWidth(ScupProgressBar.WRAP_CONTENT);

		// Set color of progressbar
		progressBar.setBackgroundImage(R.drawable.blue);

		// Sets the maximum/minimum value of progress bar.
		progressBar.setMax(200);
		progressBar.setMin(0);

		// Sets the current progress to the specified value.
		progressBar.setProgress(10);

		setActionButtonClickListener(new ActionButtonClickListener() {

			@Override
			public void onClick(ScupDialog dialog) {

				if (progressStatus <= 180) {
					progressStatus += 20;
				}
				progressBar.setProgress(progressStatus);
				update();
			}
		});
```

```
            showToast("Click action button",
                        ScupDialog.TOAST_DURATION_LONG);

            // Show progress bar
            progressBar.show();

            setBackPressedListener(new BackPressedListener() {

                  @Override
                  public void onBackPressed(ScupDialog arg0) {

                        finish();
                  }
            });
      }
}
```

To add a progress bar to the dialog:

1. Create a new ScupProgressBar with a specific style.

```
            progressBar = new ScupProgressBar(this,
                                          ScupProgressBar.STYLE_COLOR);
```

   The ScupProgressBar class supports the following styles:

Table 10: Supported types

Type	Value	Description
STYLE_COLOR	0	The progress bar is displayed in color.
STYLE_IMAGE	1	The progress bar is displayed with an image.

2. Set the width and height of the progress bar.

   Note: If the width and height are positive values between 1 and 100, they refer to a percentage of the screen width or screen height. For example, if you use `setWidth(50)`, the width of the widget is half of the screen width.

```
            progressBar.setHeight(ScupProgressBar.WRAP_CONTENT);
            progressBar.setWidth(ScupProgressBar.WRAP_CONTENT);
```

3. Set the slider padding area.

```
            progressBar.setPadding(2, 5, 2, 5);
```

4. Set the maximum and minimum values of the progress bar.

```
progressBar.setMax(200);
progressBar.setMin(0);
```

5. Set the current progress to the specified value.

```
progressBar.setProgress(10);
```

6. Show the progress bar.

```
// Show progressbar
progressBar.show();
```

4.8. Using Sliders

The ScupSlider class represents a slider widget, which works like SeekBar in Android. To use the slider:

1. Create a subclass of the ScupDialog class.
2. Add the ScupSlider widget to the application.

The following figure and code example illustrate the SliderDialog sample application that creates a ScupSlider instance in the dialog with some slider-specific attributes.

Figure 21: ScupSlider

```
public class SliderDialog extends ScupDialog{

	public SliderDialog(Context context) {
		super(context);
		// TODO Auto-generated constructor stub
	}

	@Override
	protected void onCreate() {
		// TODO Auto-generated method stub
		super.onCreate();

		setBackEnabled(true);

		ScupLabel label = new ScupLabel(this);
		label.setWidth(ScupLabel.FILL_DIALOG);
		label.setHeight(ScupLabel.WRAP_CONTENT);
		label.setAlignment(ScupLabel.ALIGN_CENTER);
		label.setText("Brightness");
		label.show();
```

```
            ScupSlider slider = new ScupSlider(this, ScupSlider.STYLE_HORIZONTAL);
            slider.setWidth(ScupSlider.FILL_DIALOG);
            slider.setHeight(ScupSlider.WRAP_CONTENT);
            slider.setPadding(2, 0, 2, 0);
            slider.setMaxValue(200);
            slider.setMinValue(0);
            slider.setStep(1);
            slider.setChangeListener(new ChangeListener() {

                  @Override
                  public void onChanged(ScupSlider slider, int value) {
                        // TODO Auto-generated method stub
                        showToast("Brightness change " + value,
                                    ScupDialog.TOAST_DURATION_LONG);
                  }
            });

            slider.show();

            setBackPressedListener(new BackPressedListener() {

                  @Override
                  public void onBackPressed(ScupDialog arg0) {
                        // TODO Auto-generated method stub
                        finish();
                  }
            });


      }

}
```

4.9. Using Thumbnail Lists

The ScupThumbnailListBox class represents a thumbnail list widget, which is a kind of extended widget. If a device cannot support this widget, it can display a base widget instead. To use the thumbnail list:

1. Create a subclass of the ScupDialog class.
2. Add the ScupThumbnailListBox widget to the application.

The following figure and code example illustrate the ThumbnailListBoxDialog sample application that creates a ScupThumbnailListBox instance in the dialog with some thumbnail list-specific attributes.

Figure 22: **ScupThumbnailListBox**

```
public class ThumbnailListBoxDialog extends ScupDialog {

	private Context mContext;
	private ScupThumbnailListBox thumbnailListBox;

	public ThumbnailListBoxDialog(Context context) {
		super(context);
		mContext = context;
	}

	@Override
	protected void onCreate() {
		super.onCreate();

		thumbnailListBox = new ScupThumbnailListBox(this);

		// Set background color for thumbnaillistbox
		thumbnailListBox.setBackgroundColor(Color.TRANSPARENT);

		// Add items to thumbnaillistbox
		thumbnailListBox.addItem(0, R.drawable.zero);
		thumbnailListBox.addItem(1, R.drawable.one);
		Bitmap bitmap = BitmapFactory.decodeResource(mContext.getResources(),
				R.drawable.two);
		thumbnailListBox.addItem(2, bitmap);
		// thumbnailListBox.addItem(2, R.drawable.two);
		thumbnailListBox.addItem(3, R.drawable.three);
		thumbnailListBox.addItem(4, R.drawable.four);
		thumbnailListBox.addItem(5, R.drawable.five);
		thumbnailListBox.addItem(6, R.drawable.six);
		thumbnailListBox.addItem(7, R.drawable.seven);
		thumbnailListBox.addItem(8, R.drawable.eight);

		// Remove Item from thumbnaillistbox
		thumbnailListBox.removeItem(7);

		// Set focusboder color
		thumbnailListBox.setFocusBorderColor(Color.RED);

		// Set item focused
		thumbnailListBox.setFocusItem(1);

		// Check scrollable of thumbnaillistbox
		if (thumbnailListBox.isScrollBarEnabled()) {
			showToast("thumbnaillistbox is scrollable",
					ScupDialog.TOAST_DURATION_SHORT);
		} else {
			showToast("thumbnaillistbox is not scrollable",
```

```
                                ScupDialog.TOAST_DURATION_SHORT);
            }

            thumbnailListBox.setBackgroundImage(R.drawable.blue);

            // Set Listener to thumbnaillistbox
            thumbnailListBox.setClickListener(new ClickListener() {

                  @Override
                  public void onClick(ScupThumbnailListBox list, int id) {
                        showToast("Item " + id + " is clicked",
                                    ScupDialog.TOAST_DURATION_SHORT);
                  }
            });

            // Show thumbnaillistbox
            thumbnailListBox.show();

      }

      @Override
      protected void onDestroy() {
            super.onDestroy();

            if (thumbnailListBox != null) {
                  thumbnailListBox.destroy();
                  thumbnailListBox = null;
            }
      }
}
```

To add a thumbnail list to a dialog:

1. Create a new ScupThumbnailListBox instance.

```
      @Override
      protected void onCreate() {
            super.onCreate();

            // Create a thumbnail ListBox.
            thumbnailListBox = new ScupThumbnailListBox(this);
```

2. Add items to the thumbnail list.

```
            // Add items to thumbnaillistbox
            thumbnailListBox.addItem(0, R.drawable.zero);
            thumbnailListBox.addItem(1, R.drawable.one);
            Bitmap bitmap = BitmapFactory.decodeResource(mContext.getResources(),
                        R.drawable.two);
            thumbnailListBox.addItem(2, bitmap);
            // thumbnailListBox.addItem(2, R.drawable.two);
            thumbnailListBox.addItem(3, R.drawable.three);
            thumbnailListBox.addItem(4, R.drawable.four);
            thumbnailListBox.addItem(5, R.drawable.five);
            thumbnailListBox.addItem(6, R.drawable.six);
            thumbnailListBox.addItem(7, R.drawable.seven);
```

```
thumbnailListBox.addItem(8, R.drawable.eight);
```

3. Remove an Item from the thumbnail list.

```
// Remove an Item to thumbnaillistbox
thumbnailListBox.removeItem(7);
```

4. Set a background image for the thumbnail list with the `setBackgroundImage()` method. You can also use the `setFocusItem()` method to set a specific list item in focus.

```
thumbnailListBox.setBackgroundImage(R.drawable.blue);
```

5. Set a listener for the thumbnail list, which is called when the user clicks a list item.

```
// Set Listener to thumbnaillistbox
thumbnailListBox.setClickListener(new ClickListener() {

      @Override
      public void onClick(ScupThumbnailListBox list, int id) {


            showToast("Item " + id + " is clicked",
                        ScupDialog.TOAST_DURATION_SHORT);
      }
});
```

6. Show the ScupThumbnailListBox widget.

```
// Show thumbnaillistbox
thumbnailListBox.show();
```

4.10. Using Time Pickers

The ScupTimePicker class represents a time picker widget, which is a view for selecting the time of day in the 12-hour or 24-hour mode. Vertical spinners control the hours, minutes, and AM or PM (in the 12-hour mode). To select the hour and minute, the user can tap the up or down symbols of the time picker. To use the time picker:

1. Create a subclass of the ScupDialog class.
2. Add the ScupSlider widget to the application.

The following figure and code example illustrate the ScupTimePickerSample application that creates a ScupTimePicker instance in the dialog with some time picker-specific attributes.

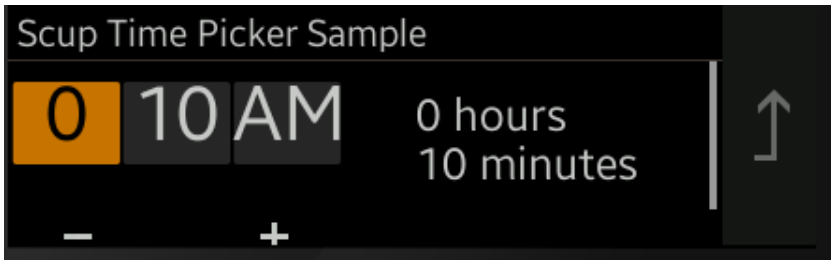

Figure 23: ScupTimePicker

```
public class ScupTimePickerSample extends ScupDialog {

	int minute = 0;
	int hour = 0;
	boolean ischanged = false;
	Handler mHandler = new Handler();
	ScupTimePicker mPicker;
	ScupLabel mLabel;

	public ScupTimePickerSample(Context context) {
		super(context);
	}

	@Override
	protected void onCreate() {
		super.onCreate();
		timeSet();
		this.setTitle("Scup Time Picker Sample");
		setAnimationType(ANIMATION_ALPHA);
	}

	private void timeSet() {
		// TODO Auto-generated method stub

		// Create a new ScupTimerPicker;
		mPicker = new ScupTimePicker(this);
		// Create a new ScupLabel
		mLabel = new ScupLabel(this);
		mLabel.setText("0 hours \n10 minutes ");
		mLabel.setPadding(10, 5, 0, 0);

		mPicker.setHeight(ScupTimePicker.WRAP_CONTENT);
		mPicker.setWidth(ScupTimePicker.WRAP_CONTENT);
		// Init time
		mPicker.setCurrentTime(0, 10, false);
		// Set a listener for status of button to start count
		mPicker.setTimeChangeListener(new ScupTimePicker.TimeChangeListener() {

		@Override
		public void onChanged(ScupTimePicker arg0, int hour, int minutes) {

			mLabel.setText(hour + " hours \n" + minutes + " minutes ");
			mLabel.setTextColor(Color.WHITE);
			mLabel.setTextSize(6);
			update();
			}
		});
		mPicker.show();
```

```
                mLabel.show();
        }

        @Override
        public void setBackPressedListener(BackPressedListener listener) {
                super.setBackPressedListener(listener);
                finish();
        }
}
```

Set the data for the Time Picker Sample with the `setCurrentTime(int hour, int minute, boolean mode)` method:

- **hour**: The hour.
- **minute**: The minute.
- **mode**: The mode. If the parameter is true, the 24-hour mode is used. Otherwise, the 12-hour mode is used.

```
        mPicker.setCurrentTime(0, 10, false);
```

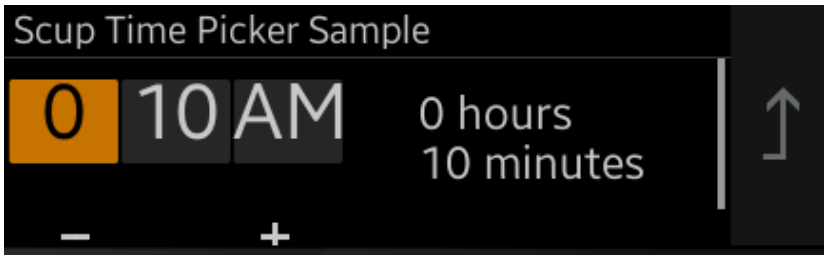

Figure 24: TimePicker in 12-hour mode

```
        mPicker.setCurrentTime(0, 10, true);
```

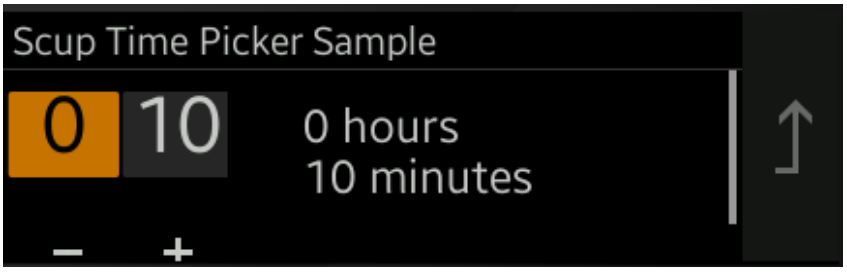

Figure 25: TimePicker in 24-hour mode

When the Time Picker is used as an alarm, it has to count down the time. The `getCurrentMinute()` and `getCurrentHour()` methods return the current minute and hour respectively. This value can be different on the device side, and synchronizes data at specific times. The time set is saved in the `hour` and `minute` variables.

```
        minute = mPicker.getCurrentMinute();
        hour = mPicker.getCurrentHour();
```

Set the time change listener to track changes in time. In this listener, the button is set on and can be clicked to count down.

```
mPicker.setTimeChangeListener(new ScupTimePicker.TimeChangeListener() {

        @Override
        public void onChanged(ScupTimePicker arg0, int hour, int minutes) {

                mLabel.setText(hour + " hours \n" + minutes + " minutes ");
                mLabel.setTextColor(Color.WHITE);
                mLabel.setTextSize(6);
                update();
        }
});
```

4.11. Using Date Pickers

The ScupDatePicker class represents a date picker widget, which is a kind of extended widget for selecting the date. If a device cannot support this widget, it can display a base widget instead. To use the date picker:

1. Create a subclass of the ScupDialog class.
2. Add the ScupSlider widget to the application.

The following figure and code example illustrate the Datepicker_Dialog sample application that creates a ScupDatePicker instance in the dialog with some date picker-specific attributes.

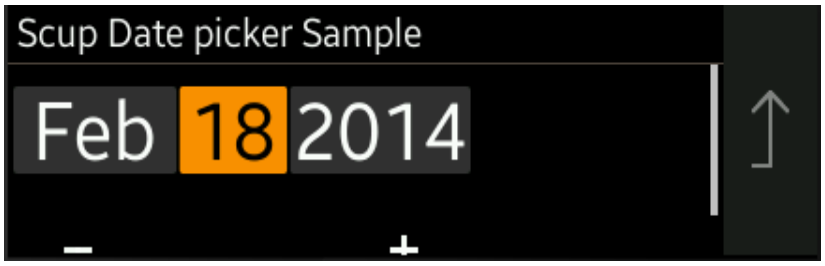

Figure 26: ScupDatePicker

```
public class Datepicker_Dialog extends ScupDialog {

        private ScupDatePicker scupDatePicker;

        public Datepicker_Dialog(Context context) {
                super(context);
                // TODO Auto-generated constructor stub
        }

        @Override
        protected void onCreate() {
                // TODO Auto-generated method stub
```

```
        super.onCreate();

        setBackEnabled(true);

        // Create new instance of ScupDatePicker
        scupDatePicker = new ScupDatePicker(this,
                    ScupDatePicker.STYLE_YEAR_MONTH_DAY);
        // Set current date for DatePicker
        scupDatePicker.setCurrentDate(2, 15, 2014);
        scupDatePicker.setMinDate(2, 15, 2014);

        String[] strings = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
                    "Aug", "Sep", "Oct", "Nov", "Dec" };
        // Set month text
        scupDatePicker.setMonthExpression(strings);
        scupDatePicker.show();
        // Set an event for ScupDatepicker
        scupDatePicker
              .setDateChangeListener(new ScupDatePicker.DateChangeListener() {

              @Override
              public void onChanged(ScupDatePicker arg0, int arg1,
                          int arg2, int arg3) {
              // TODO Auto-generated method stub
              // In this sample, I simply set the current
              //                    and show it as a toast
              String string = "Month: " + arg1 + ", Day:" + arg2
                                        + ", Year:" + arg3;

              Datetpicker_Dialog.this.showToast(string,
                                        ScupDialog.TOAST_DURATION_LONG);
              }
              });

        // Set event when user press back button on the Wingtip device, in this
        // sample, I simply finish the dialog
        setBackPressedListener(new BackPressedListener() {

              @Override
              public void onBackPressed(ScupDialog arg0) {
                    // TODO Auto-generated method stub
                    finish();
              }
        });

        // Set some attributes for dialog
        this.setTitle("Date picker wingtip");
        this.setWidgetGap(5);
  }

  @Override
  protected void onDestroy() {
        // TODO Auto-generated method stub

        // Destroy all widgets for avoid memory leak
        scupDatePicker.destroy();
       super.onDestroy();

  }
```

```
}
```

To add a date picker to a dialog:

1. Call the ScupDatePicker class constructor. Inside the constructor, pass the dialog object that contains the ScupDatePicker, and the date picker style.

```
// create new instance of ScupDatePicker
scupDatePicker = new ScupDatePicker(this,ScupDatePicker.STYLE_YEAR_MONTH_DAY);
```

The ScupDatePicker class supports the following styles:

Table 11: ScupDatePicker styles

Type	Value	Description
STYLE_MONTH_DAY_YEAR	0	The month information is displayed first.
STYLE_YEAR_MONTH_DAY	1	The year information is displayed first.

2. Set the current and minimum date for the date picker.

```
// set current date for DatePicker
scupDatePicker.setCurrentDate(2, 15, 2014);
scupDatePicker.setMinDate(2, 15, 2014);
```

3. Set the month text expression for the date picker.

Note: In the `setMonthExpression()` method, if you do not set this attribute, the default month text expressions are "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", and "Dec".

```
String[] strings = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
                     "Aug", "Sep", "Oct", "Nov", "Dec" };
// Set month text
scupDatePicker.setMonthExpression(strings);
```

4. Add an event for the date picker, if needed. Call `setDateChangeListener()` when the current date showed on the widget is changed, and pass an instance of ScupDatePicker.DateChangeListener listener to it. Inside the listener, override the `onChanged()` method.

```
// Set an event for ScupDatepicker
scupDatePicker
.setDateChangeListener(new ScupDatePicker.DateChangeListener() {
@Override
       public void onChanged(ScupDatePicker arg0, int arg1,
              int arg2, int arg3) {
// TODO Auto-generated method stub
// In this sample, I simply show it as a toast
String string = "Month: " + arg1 + ", Day:" + arg2
                                          + ", Year:" + arg3;

Datetpicker_Dialog.this.showToast(string,
                                          ScupDialog.TOAST_DURATION_LONG);
```

```
            }
      });
```

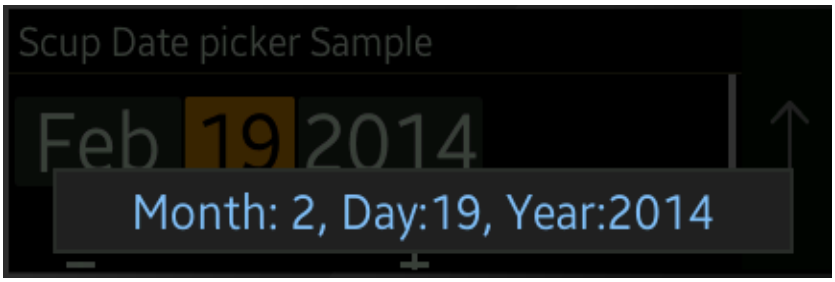

Figure 27: DatePicker - DateChangeListener

4.12. Using Media Controllers

The ScupMediaController class represents a media controller widget, which is a kind of extended widget. If a device cannot support this widget, it can display a ScupButton widget instead. To use the media controller:

1. Create a subclass of the ScupDialog class.
2. Add the ScupMediaController widget to the application.

The following figure and code example illustrate the ScupMediaControllerDemo sample application that creates a ScupMediaController instance in the dialog with some media controller-specific attributes.

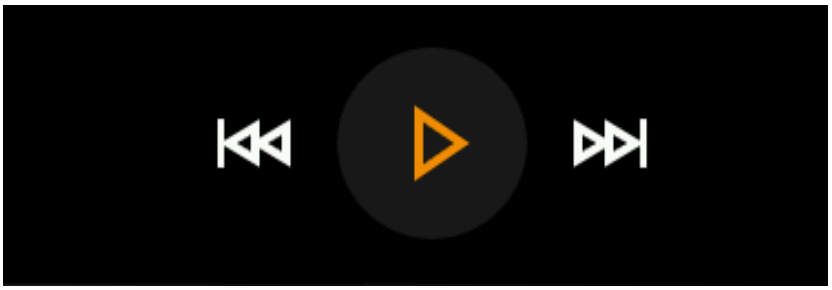

Figure 28: ScupMediaController

```
public class ScupMediaControllerDemo extends ScupDialog {

      private ScupMediaController scupMedia;

      public ScupMediaControllerDemo(Context context) {
            super(context);
      }

      @Override
      protected void onCreate() {
            super.onCreate();

            setBackEnabled(true);

            scupMedia = new ScupMediaController(this);
            scupMedia.setMediaState(ScupMediaController.MEDIA_STATE_PAUSE);
            scupMedia.setPadding(25, 5, 7, 0);
```

```
            scupMedia.setClickListener(new ScupMediaController.ClickListener() {
            @Override
            public void onClick(ScupMediaController arg0, int arg1) {
            if(scupMedia.getMediaState() == ScupMediaController.MEDIA_STATE_PLAY)
                  scupMedia.setMediaState(ScupMediaController.MEDIA_STATE_PAUSE);

            else if(scupMedia.getMediaState() == ScupMediaController.MEDIA_STATE_PAUSE)
                  scupMedia.setMediaState(ScupMediaController.MEDIA_STATE_PLAY);
            if(arg1 == ScupMediaController.BUTTON_TYPE_NEXT){
                  scupMedia.setMediaState(ScupMediaController.MEDIA_STATE_PLAY);
                  showToast("NEXT", Toast.LENGTH_LONG);
            }
            if(arg1 == ScupMediaController.BUTTON_TYPE_PREV){
                  scupMedia.setMediaState(ScupMediaController.MEDIA_STATE_PLAY);
                  showToast("PREV", Toast.LENGTH_LONG);
                  }
            }
      });

      scupMedia.show();
      }

      @Override
      public void finish() {
            super.finish();
            if (scupMedia != null) {
                  scupMedia.destroy();
                  scupMedia = null;
            }
      }

      @Override
      protected void onDestroy() {
            super.onDestroy();
            finish();
      }
}
```

To add a media controller to a dialog:

1. Create a class extending the ScupDialog base and the constructor for this class.

```
public class ScupMediaControllerDemo extends ScupDialog {

      private ScupListBox mListBox;
      public ScupMediaControllerDemo(Context context) {
            super(context);
      }
```

2. Override the `onCreate()` method from the base class.

```
protected void onCreate() {
      super.onCreate();
//...
```

3. Create a new ScupMediaController instance.

```
final ScupMediaController scupMedia = new ScupMediaController(this);
            scupMedia.setMediaState(ScupMediaController.MEDIA_STATE_PAUSE);
            scupMedia.setPadding(25, 5, 7, 0);
```

4. Set and track the state of the media controller:

    - o `getMediaState()`: Returns the current state as MEDIA_STATE_PLAY or MEDIA_STATE_PAUSE.
    - o `setMediaState(int state)`: Sets the state as MEDIA_STATE_PLAY or MEDIA_STATE_PAUSE.

In the widget, the user can click the play, pause, next, and prev buttons to change the media state.

```
if(btPlay.getMediaState() == ScupMediaController.MEDIA_STATE_PLAY)
      btPlay.setMediaState(ScupMediaController.MEDIA_STATE_PAUSE);
else if(btPlay.getMediaState() == ScupMediaController.MEDIA_STATE_PAUSE)
       btPlay.setMediaState(ScupMediaController.MEDIA_STATE_PLAY);

if(arg1 == ScupMediaController.BUTTON_TYPE_NEXT)
      showToast("NEXT", Toast.LENGTH_LONG);
if(arg1 == ScupMediaController.BUTTON_TYPE_PREV)
      showToast("PREV", Toast.LENGTH_LONG);
```

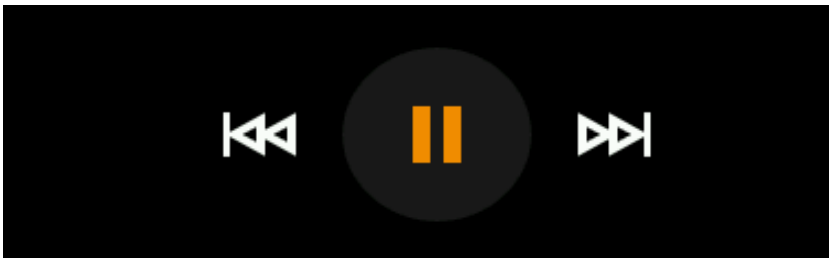

Figure 29: Media Controller pause button

Figure 30: Media Controller next button

Figure 31: Media Controller **prev button**

Copyright

Though every care has been taken to ensure the accuracy of this document, Samsung Electronics Co., Ltd. cannot accept responsibility for any errors or omissions or for any loss occurred to any person, whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

Samsung Electronics Co. Ltd. may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

No part of this document may be communicated, distributed, reproduced or transmitted in any form or by any means, electronic or mechanical or otherwise, for any purpose, without the prior written permission of Samsung Electronics Co. Ltd.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

For more information, please visit http://developer.samsung.com/