keynote

DeviceAnywhere Enterprise

Java Examples

DeviceAnywhere Enterprise 6.2.1

DeviceAnywhere Enterprise Automation 6.2.1

April 2014

Copyright Notice

Please forward any comments or suggestions regarding this document to Keynote Support.

Keynote Systems, Inc.
777 Mariners Island Blvd.
San Mateo, CA 94404

Contents

1 About This Document

This document provides sample Java code and brief definitions of DeviceAnywhere Enterprise Java API functions for:

- Basic device interactions such as swipes and changing orientation
- Text-based reference points—verifying that text on the device screen matches a string of reference text—when defining a text-based reference point, text from the device screen is transformed using one of several methods to facilitate recognition.
- Image-based reference points—verifying that an image region on the device screen matches a reference image
- Advanced operations based on text recognition such as touching a string of text and extracting it to a variable
- Variables and parameters—passing parameters and using and setting the value of project (global) variables
- Other operations such as specifying dependent JAR files, waiting for a reference point defined in a state, collecting script execution return values and device screen proofs, and breaking execution when a step fails.

This document assumes that you are familiar with the Java programming language and the following operations in DeviceAnywhere Studio:

- Interacting with devices
- Using visual scripting commands
- Scripting entities—actions, states, test cases, test cycles

The table below describes the typographical conventions used in DeviceAnywhere documentation.

Style	Element	Example
Blue	Links and email addresses	http://www.keynote.com
Bold	User interface elements such as menu items	Click **My Devices** in DeviceAnywhere Studio.
`Monospace`	Commands, code output, filenames, directories	Right-click the project's `test cases` directory.
`Monospace bold`	User input	In a command window, type **`adb devices`**.
Italic	Document titles and emphasis	Refer to the *DeviceAnywhere Enterprise Private System Installation Guide* for instructions on setting up server infrastructure.

Contacting Support

If you have any comments or suggestions regarding this document, contact Keynote Support at http://support.keynote.com or 1-888-KEY-SYST (539-7978).

Additional Documentation

You can find additional documentation at:

- DeviceAnywhere Enterprise shared system with private devices: http://www.keynotedeviceanywhere.com/dae-pvt-devices-documentation.html
- DAE Automation: http://www.keynotedeviceanywhere.com/testing-automation-documentation.html

In addition, you can access documentation from the **Help** menu in DeviceAnywhere Studio.

2 Using the Java API

DeviceAnywhere Studio offers several options for writing scripts that run on live DeviceAnywhere Enterprise devices:

- The visual interface in DeviceAnywhere Studio offers a UI-based approach with recordable and/or drag-and-drop commands that make up a script.
- The second approach is to write scripts in Java, using the Java API provided by Keynote. Users can utilize DeviceAnywhere Studio's built-in Java editor or any other editor, such as Eclipse.

Writing mobile automation scripts in Java typically requires specific information about device interaction, obtainable from visual commands in DeviceAnywhere Studio. Some examples of this are:

- Performing a swipe—you need swipe coordinates (start and end points and the number of intermediate steps) captured in a command or device interaction log of DeviceAnywhere Studio.
- Text matching with transformation—text transformation aims to render device screen text as dark text on a light background or vice versa, which are easiest to recognize. Choosing the appropriate transformation method requires the visual UI for trial and error of different settings. Additionally, transformation values—transformation color and tolerance level—must be obtained from DeviceAnywhere Studio.
- Touching an area that is offset from a reference image or text string (clicking on something next to the image or text) – requires the x or y offset of the area to touch from the reference image or text.

While DeviceAnywhere Enterprise users can easily program in Java, they will also want to launch DeviceAnywhere Studio and acquire the device they are scripting on, so they can easily find values needed for scripts.

In the examples below, comments explain lines of code and point out when certain values must be obtained from DeviceAnywhere Studio.

3 Examples

In the sections below, like examples DeviceAnywhere Enterprise Java code are grouped together. Each example contains comments to help understand the code. Examples contain sample or dummy values with suggestions for getting values from DeviceAnywhere Studio. Sample values must be replaced by values relevant to your testing.

Main API functions used are described briefly above each example. Consult the DAE Automation Java API documentation for details.

3.1 Basic Device Interaction

You can perform basic device interaction (key presses, swipes) with the `sendKeys()` function. This function takes as arguments the keys to be pressed or the coordinates to be touched and the key mode. Optionally, it also takes the `KeysHelper()` object as an argument, which you can use to specify hold and delay times.

Hold time is the length of time, in milliseconds, for which you press a key or coordinate. Delay time is the time, in milliseconds, between key presses or touches.

The key mode specifies the type of data that can be entered into a given field on the device.

3.1.1 Performing a Swipe

```
//// How to use DeviceAnywhere Studio to implement a swipe in Java:
// 1. Drag a new Send Keys command into a visual action.
// 2. Right-click Send Keys > "Start Recording"
// 3. Perform a swipe: Shift/Ctrl key + drag mouse across device screen.
// 4. Right-click Send Keys > "Stop Recording"
// 5. Open the Send Keys command and copy swipe coordinates from the "Text to
Send" field.
// 6. Paste it into an API call as follows:
device.sendKeys("[Swipe(554,277)(124,237)(10)]", KeyMode.ALPHA);

// 7. This example requires the following imports to your Java code:
import com.mc.api.device.helper.KeyMode;

// Understanding swipe coordinates:
// [Swipe(x1,y1)(x2,y2)(z)],
// x1, y1 are coordinates of swipe start position.
// (x2, y2) represents swipe end coordinates.
// z represents number of intermediate steps between swipe start and end.
```

NOTE You need to use a hold time of 2000ms for a swipe on software-integrated devices. See Doing a Long Key Press below for information on creating a `KeysHelper()` object to set hold time. `sendKeys()` takes the `KeysHelper()` object as an argument.

3.1.2 Doing a Long Key Press

```
//// How to perform a long key press, e.g., touching the coordinate (398,139)
for 2000ms (2 seconds)
// 1. Create a KeysHelper object to specify hold time (2000 ms).
// Default hold time is 200ms.
```

```
KeysHelper kh = new KeysHelper();
kh.setHoldTime(2000);

// 2. Pass the KeysHelper object and touch coordinates to SendKeys arguments.
device.sendKeys("[Touch(398,139)", KeyMode.ALPHA, kh);

//This example requires the following imports to your Java code:
import com.mc.api.device.helper.KeyMode;
import com.mc.api.device.helper.KeysHelper;
```

3.1.3 Hardware Commands

Use the `setHardwareState()` function to perform hardware operations on a device, e.g., change the orientation, attach/detach the data cable, or turn the camera light on/off. `setHardwareState()` takes the `HardwareHelper()` object as an argument (with subclasses for different hardware operations, e.g., `CameraLightHelper()`).

```
//// Example: Hardware Commands

// 1 – Set Device Orientation
device.setOrientation(Orientation.CLOSED_RIGHT);
//or   device.setOrientation(Orientation.CLOSED_LEFT);
//or   device.setOrientation(Orientation.CLOSED_VERTICAL);

// 2 - Data Cable
DataCableHelper dch = new DataCableHelper();
dch.setDataCableState(true);
// or dch.setDataCableState(false);
// true = attach cable, false = detach cable
device.setHardwareState(dch);

// 3 - Camera Light
CameraLightHelper clh = new CameraLightHelper();
clh.setCameraLightState(true);
// or clh.setCameraLightState(false);
// true = turn on camera light, false = turn off camera light
device.setHardwareState(clh);
```

3.2 Text Matching

Text matching is one of the ways available in DeviceAnywhere Enterprise to use device conditions or output to verify a sequence of device interactions. In text matching, you use a string of text from a device screen to define a *text-based reference point*. DeviceAnywhere Enterprise uses optical character recognition (OCR) technology to verify that text found on the device screen during run time matches the reference string.

A reference point serves as an expected result against which the outcome of a script can be verified. For instance, you can use static text from an application to verify that a device has opened it correctly.

When using a device screen to define reference text, you must facilitate text recognition by *transforming the text*. To transform text in DeviceAnywhere Enterprise you can use one of the following options:

- Select the color of text for the transformation engine to extract.
- Specify the background color of the area from which the transformation engine should extract text.

- Convert a colored device screen image to black and white.
- Adjust the contrast of the device screen.

The DAE Automation User Guide discusses text-based reference points and transformation methods in detail, including the transformation methods best suited to different types of device screens.

The following examples describe how to set up a text-based reference point in Java. Each example uses a different transformation method. Text verification requires the `waitText()` function, which takes the text string to wait for, time to wait (in milliseconds), and the `TextHelper()` object as arguments. The `TextHelper()` object, in turn, has subclasses for defining each transformation method.

3.2.1 Matching Text Based on Text (Foreground) Color

When using text color-based transformation, you must specify the color of text and a threshold value for variations in the color specified. These values are best obtained from a command, e.g., Wait Text, in DeviceAnywhere Studio.

1. In the command, capture the device screen containing the string of text you wish to wait for.
2. Then select the color of the text you wish to wait for. Hover over the color selector to see the RBG component values of the selected color.
3. Next, set the threshold value (scale: 0 – 128). The threshold value is added to and subtracted from the RGB component values of the text color.

```
//// Wait for text, using foreground color to transform the device screen.

// 1. Construct a TextHelper object using foreground (text) color.
/*
The first parameter is the RGB component values of the color of text to
extract.
The second is the threshold value for variations in the color of extracted
text (0 – 128).
The best way to obtain these values is to create a Wait Text command in
DeviceAnywhere Studio, selecting the foreground color, threshold, and
optionally, screen region in which to look for text.
Mouse over the color and threshold controls to view corresponding values.
*/
final TextHelper th = new TextForegroundHelper( new Color(248,252,248), 37 );

// 2. Optionally, set the TextHelper's canvas region.
// (x coordinate of top-left corner, y coordinate, width, height)
th.setCanvasRegion( 0, 0, 100, 100 );

// 3. Set the time to wait for the text (in milliseconds).
final int    timeout       = 5000;

// 4. Set the text you wish to wait for.
final String textToWaitFor = "Sample text string";

// 5. Call the Device.waitText function with the parameters specified above.
WaitMatch wm = device.waitText(textToWaitFor, timeout, th);

// 6. Check to see if text was found and branch accordingly.
if ( wm.isMatched() ) {
      // 6a. Text was found, do something...
} else {
      // 6b. Text was not found, do something else...
}

// The following imports will need to be added to your Java code:
import com.mc.api.device.helper.TextForegroundHelper;
import com.mc.api.device.helper.TextHelper;
import com.mc.api.device.helper.WaitMatch;
```

3.2.2 Waiting for Text Based on Background Color

When using background color-based transformation, you must specify the background color of text to extract and a threshold value for variations in the color specified. These values are best obtained from a command, e.g., Wait Text, in DeviceAnywhere Studio.

1 In the command, capture the device screen containing the string of text you wish to wait for.

2 Then select the background color of the text you wish to wait for. Hover over the color selector to see the RBG component values of the selected color.

3 Next, set the threshold value (scale: 0 – 128). The threshold value is added to and subtracted from the RGB component values of the text color.

```
//// Wait for text, using background color to transform the device screen.

// 1. Construct a TextHelper object using background color.
/*
The first parameter is the RGB component values of the background color of
text to extract.
The second is the threshold value for variations in the background color of
extracted text (0 – 128).
The best way to obtain these values is to create a Wait Text command in
DeviceAnywhere Studio, selecting the background color, threshold, and
optionally, screen region in which to look for text.
Mouse over the color and threshold controls to view corresponding values.
*/
final TextHelper th = new TextBackgroundHelper( new Color(248,252,248), 37 );

// 2. Optionally, set the TextHelper's canvas region.
// (x coordinate of top-left corner, y coordinate, width, height)
th.setCanvasRegion( 0, 0, 100, 100 );

// 3. Set the time to wait for the text (in milliseconds).
final int    timeout       = 5000;

// 4. Set the text you wish to wait for.
final String textToWaitFor = "Sample text string";

// 5. Call the Device.waitText function with the parameters specified above.
WaitMatch wm = device.waitText(textToWaitFor, timeout, th);

// 6. Check to see if text was found and branch accordingly.
if ( wm.isMatched() ) {
      // 6a. Text was found, do something...
} else {
      // 6b. Text was not found, do something else...
}
```

```
// The following imports will need to be added to your Java code:
import com.mc.api.device.helper.TextBackgroundHelper;
import com.mc.api.device.helper.TextHelper;
import com.mc.api.device.helper.WaitMatch;
```

3.2.3 Waiting for Text by Transforming to Black and White

When using transformation to black and white, you must specify the dividing color between black and white and a threshold value for variation in the color specified. These values are best obtained from a command, e.g., Wait Text, in DeviceAnywhere Studio.

1 In the command, capture the device screen containing the string of text you wish to wait for.

2 Then select the dividing color—darker colors are mapped to black and lighter colors are mapped to white. Hover over the color selector to see the RBG component values of the selected color.

3 Next, adjust the selected (scale: -128 – 128) color if you are not able to pick exactly the color you want. The slider value is added to the RGB component values of the selected color. A negative slider value makes the selected color darker. A positive slider value makes the selected color lighter. Leave the slider at 0 if you are satisfied with the dividing color you have selected.

```
//// Wait for text, transforming the device screen to black and white.

// 1. Construct a TextHelper object using black and white transformation.
/*
The first parameter is the RGB component values of the dividing color between
black and white.
The second adjusts the value of the dividing color (-128 – 128).
The best way to obtain these values is to create a Wait Text command in
DeviceAnywhere Studio, selecting the dividing color, slider value, and
optionally, screen region in which to look for text.
Mouse over the color and slider controls to view corresponding values.
*/
final TextHelper th = new TextBlackWhiteHelper( new Color(248,252,248), 37 );

// 2. Optionally, set the TextHelper's canvas region.
// (x coordinate of top-left corner, y coordinate, width, height)
th.setCanvasRegion( 0, 0, 100, 100 );

// 3. Set the time to wait for the text (in milliseconds).
final int    timeout       = 5000;

// 4. Set the text you wish to wait for.
final String textToWaitFor = "Sample text string";

// 5. Call the Device.waitText function with the parameters specified above.
WaitMatch wm = device.waitText(textToWaitFor, timeout, th);

// 6. Check to see if text was found and branch accordingly.
if ( wm.isMatched() ) {
      // 6a. Text was found, do something...
} else {
      // 6b. Text was not found, do something else...
}

// The following imports will need to be added to your Java code:
```

```
import com.mc.api.device.helper.TextBlackWhiteHelper;
import com.mc.api.device.helper.TextHelper;
import com.mc.api.device.helper.WaitMatch;
```

3.2.4 Waiting for Text by Adjusting Contrast

When transforming by adjusting contrast, you must specify the color with respect to which contrast is adjusted and a value for contrast adjustment. These values are best obtained from a command, e.g., Wait Text, in DeviceAnywhere Studio.

1 In the command, capture the device screen containing the string of text you wish to wait for.

2 Then select the delimiting color—darker colors are darkened while lighter colors are lightened. Hover over the color selector to see the RBG component values of the selected color.

3 Next, adjust contrast (scale: 0 – 100) to the extent desired.

```
//// Wait for text, adjusting contrast of the device screen.

// 1. Construct a TextHelper object using contrast adjustment.
/*
The first parameter is the RGB component values of the delimiting color.
The second adjusts contrast (0 – 100).
The best way to obtain these values is to create a Wait Text command in
DeviceAnywhere Studio, selecting the delimiting color, contrast value, and
optionally, screen region in which to look for text.
Mouse over the color and slider controls to view corresponding values.
*/
final TextHelper th = new TextAdjustContrastHelper( new Color(248,252,248),
37 );

// 2. Optionally, set the TextHelper's canvas region.
// (x coordinate of top-left corner, y coordinate, width, height)
th.setCanvasRegion( 0, 0, 100, 100 );

// 3. Set the time to wait for the text (in milliseconds).
final int    timeout       = 5000;

// 4. Set the text you wish to wait for.
final String textToWaitFor = "Sample text string";

// 5. Call the Device.waitText function with the parameters specified above.
WaitMatch wm = device.waitText(textToWaitFor, timeout, th);

// 6. Check to see if text was found and branch accordingly.
if ( wm.isMatched() ) {
      // 6a. Text was found, do something...
} else {
      // 6b. Text was not found, do something else...
}

// The following imports will need to be added to your Java code:
import com.mc.api.device.helper.TextAdjustContrastHelper;
import com.mc.api.device.helper.TextHelper;
import com.mc.api.device.helper.WaitMatch;
```

3.3 Image Matching

Image matching is one of the ways available in DeviceAnywhere Enterprise to use device conditions or device output to verify the result of a sequence of device interactions. In image matching, you define an area of the device screen as a reference image, or an *image-based reference point*. DeviceAnywhere Enterprise uses pixel-to-pixel image matching technology to match the actual device screen to the reference image for script verification.

A reference point serves as an expected result against which the outcome of a script can be verified. For instance, you can define an application icon as an image-based reference point for the device home screen.

The DAE Automation User Guide discusses image-based reference points in detail.

Image verification requires the `waitImage()` function, which takes the time to wait (in milliseconds) for the reference image, and the `ImageDataHelper()` object as arguments. The `ImageDataHelper()` object, in turn, takes the image region to wait for as an argument.

The example below describes how to set up an image-based reference point and return the coordinates of the of the match location (generally the center of the reference image) at run time.

3.3.1 Waiting for an Image

```
/*
Defining a reference region on a device screen image located in an external
file
*/

// 1. File containing image region to be matched on device at run time
final File f = new File("myimage.png");

// 2. Read it into a BufferedImage.
final BufferedImage bi = ImageIO.read(f);

// 3. Create an ImageDataHelper object from the BufferedImage.
final ImageDataHelper idh = new ImageDataHelper(bi);

// 4. Define a region of the source image to look for.
// (x coordinate of top-left corner, y coordinate, width, height)
idh.setRegion(100,100,150,200);

// 5. Set the time to wait for the image (in milliseconds).
final int timeout = 5000;

// 6. Call waitImage with the ImageDataHelper and timeout arguments.
WaitMatch wm = device.waitImage(idh, timeout);

// 7. See if image was matched.
if ( wm.isMatched() ) {
      // 7a. Image was found, do something...
} else {
      // 7b. Image was not found, do something else...
}

// The following imports will need to be added to your Java code:
import com.mc.api.device.helper.WaitMatch;
import com.mc.api.device.helper.ImageDataHelper
```

3.3.2 Finding and Touching an Image

The `findAndTouch()` function searches for and touches the center of a specified region on a device screen. The screen region to touch is defined in much the same way as a reference image.

```
/*
Defining a region to find and touch on a device screen image located in an
external file; obtaining x, y coordinates of region touched at run time
*/

// 1. File containing image region to be found and touched at run time
final File f = new File("myimage.png");

// 2. Read it into a BufferedImage.
final BufferedImage bi = ImageIO.read(f);

// 3. Create an ImageDataHelper object from the BufferedImage.
final ImageDataHelper idh = new ImageDataHelper(bi);

// 4. Define a region of the source image to find.
// (x coordinate of top-left corner, y coordinate, width, height)
idh.setRegion(100,100,150,200);

// 5. Create a KeysHelper object to pass in hold (touch) time.
KeysHelper kh = new KeysHelper();

/*
6. Optionally, set parameters such as the touch duration to touch longer than
the default (200 ms).
*/
kh.setHoldTime( 2000 ); // 2000ms = 2 seconds

// 7. Set the time to wait to touch the image (in milliseconds).
final int timeout = 5000; // 5000ms = 5 seconds

/*
8. Call findAndTouch with ImageDataHelper, KeysHelper, and timeout arguments.
*/
WaitMatch wm = device.findAndTouch(idh, timeout, kh);

// 9. See if the screen region was found and touched.
if ( wm.isMatched() ) {
      // 9a. Image was found, do something...
} else {
      // 9b. Image was not found, do something else...
}
```

3.3.3 Obtaining Coordinates of Image Matched (or Touched)

Use the `getMatchLocation()` function (in step 7 of this example) of the `WaitMatch()` class to return the coordinates of the of the match location (generally the center of the reference image) at run time. You can also insert this step when finding and touching an image.

```
/*
Defining a reference region on a device screen image located in an external
file; obtaining x, y coordinates of region matched at run time
*/
```

```
// 1. File containing image region to be matched on device at run time
final File f = new File("myimage.png");

// 2. Read it into a BufferedImage.
final BufferedImage bi = ImageIO.read(f);

// 3. Create an ImageDataHelper object from the BufferedImage.
final ImageDataHelper idh = new ImageDataHelper(bi);

// 4. Define a region of the source image to look for.
// (x coordinate of top-left corner, y coordinate, width, height)
idh.setRegion(100,100,150,200);

// 5. Set the time to wait for the image (in milliseconds).
final int timeout = 5000;

// 6. Call waitImage with the ImageDataHelper and timeout arguments.
WaitMatch wm = device.waitImage(idh, timeout);

// 7. See if image was matched.
if ( wm.isMatched() ) {
      // 7a. If matched, obtain x, y coordinates of match location.
      Point p = wm.getMatchLocation();
      int x = p.x;
      int y = p.y;
} else {
      // 7b. If not matched, do something else...
}
```

3.4 Additional Operations Using Text Recognition

In addition to waiting for a string of text for script verification, you can find and touch a text string (as in a hyperlink), obtain the coordinates of the screen region touched/matched, and extract a text string to a variable.

3.4.1 Find and Touch Text

The `findTextAndTouch()` function searches for and touches the center of a specified text string on a device screen, e.g., a hyperlink. The text to touch is defined in much the same way as reference text, and the device screen must be *transformed* in order to aid text recognition. The example below finds and touches a string of text while transforming the device screen by adjusting contrast.

```
//// Find and touch text, transforming device screen by adjusting contrast.

// 1. Construct a TextHelper object using contrast adjustment.
/*
The first parameter is the RGB component values of the delimiting color.
The second adjusts contrast (0 – 100).
The best way to obtain these values is to create a Wait Text command in
Studio, selecting the delimiting color, contrast value, and optionally,
screen region in which to look for text.
Mouse over the color and slider controls to view corresponding values.
*/
final TextHelper th = new TextAdjustContrastHelper( new Color(248,252,248),
37 );
```

```
// 2. Optionally, set the TextHelper's canvas region.
// (x coordinate of top-left corner, y coordinate, width, height)
th.setCanvasRegion( 0, 0, 100, 100 );

// 3. Set the time to wait for the text (in milliseconds).
final int    timeout       = 5000;

// 4. Set the text you wish to find and touch.
final String textToTouch = "my text";

// 5. Construct a KeysHelper object to pass in hold (touch) time.
final KeysHelper kh = new KeysHelper();

/*
6. Optionally, set parameters such as the touch duration to touch longer than
the default (200 ms).
*/
kh.setHoldTime(2000);

/*
7. Call Device.findTextAndTouch with textToTouch, timeout, TextHelper, and
KeysHelper arguments.
*/
WaitMatch wm = device.findTextAndTouch(textToTouch, timeout, th, kh);

// 8. Check to see if text was touched and branch accordingly.

if ( wm.isMatched() ) {
      // 8a. Text was found, do something...
} else {
      // 8b. Text was not found, do something else...
}

// Java imports needed for the code sample above:
import com.mc.api.device.helper.TextAdjustContrastHelper;
import com.mc.api.device.helper.TextHelper;
import com.mc.api.device.helper.WaitMatch;
import com.mc.api.device.helper.KeysHelper;
```

3.4.2 Obtaining Coordinates of Text Matched

Use the Use the `getMatchLocation()` function (in step 6 of this example) of the `WaitMatch()` class to return the coordinates of the of the match location (generally the center of the reference text) at run time. You can also insert this step when finding and touching text.

```
/*
Wait for text, adjusting contrast of the device screen; obtain x, y
coordinates of found text.
*/

// 1. Construct a TextHelper object using contrast adjustment.
/*
The first parameter is the RGB component values of the delimiting color.
The second adjusts contrast (0 – 100).
The best way to obtain these values is to create a Wait Text command in
Studio, selecting the delimiting color, contrast value, and optionally,
screen region in which to look for text.
Mouse over the color and slider controls to view corresponding values.
*/
final TextHelper th = new TextAdjustContrastHelper( new Color(248,252,248),
37 );

// 2. Optionally, set the TextHelper's canvas region.
// (x coordinate of top-left corner, y coordinate, width, height)
th.setCanvasRegion( 0, 0, 100, 100 );

// 3. Set the time to wait for the text (in milliseconds).
final int    timeout       = 5000;

// 4. Set the text you wish to wait for.
final String textToWaitFor = "Sample text string";

// 5. Call the Device.waitText function with the parameters specified above.
WaitMatch wm = device.waitText(textToWaitFor, timeout, th);

// 6. Check to see if text was found and branch accordingly.
if ( wm.isMatched() ) {
      // 6a. If matched, obtain x, y coordinates of match location.
      Point p = wm.getMatchLocation();
      int x = p.x;
      int y = p.y;
} else {
      // 6b. If not matched, do something else...
}

// Java imports needed for the code sample above:
import com.mc.api.device.helper.TextAdjustContrastHelper;
import com.mc.api.device.helper.TextHelper;
import com.mc.api.device.helper.WaitMatch;
import java.awt.Point;
```

3.4.3 Extracting Text to a Project (Global) Variable

The `extractText()` function allows you to extract a text string from a device screen and store it in a parameter or variable for use later.

NOTE Be sure to use a variable or parameter of the appropriate type for the data you are extracting, e.g., use a variable of type **Text** to store an alphanumeric value.

The text to extract is defined in much the same way as reference text, and the device screen must be *transformed* in order to aid text recognition. The example below extracts a string of text while using text color-based transformation.

```
//// Extract text to a project/global variable.

// 1. Construct a TextHelper object using foreground (text) color.
/*
The first parameter is the RGB component values of the color of text to
extract.
The second is the threshold value for variations in the color of extracted
text (0 – 128).
```

```
The best way to obtain these values is to create a Wait Text command in
Studio, selecting the foreground color, threshold, and optionally, screen
region in which to look for text.
Mouse over the color and threshold controls to view corresponding values.
*/
final TextHelper th = new TextForegroundHelper( new Color(248,252,248), 37 );

// 2. Optionally, set the TextHelper canvas region from which to extract text.
// (x coordinate of top-left corner, y coordinate, width, height)
th.setCanvasRegion( 0, 0, 100, 100 );

// 3. Extract text from device and store in a project variable.
String extractedText = device.extractText(th);
this.getProject().setValue( "myProjectVariable", extractedText );

// Java imports needed for the code sample above:
import com.mc.api.device.helper.TextHelper
import com.mc.api.device.helper.TextForegroundHelper
```

3.5 Working with Variables and Parameters

DeviceAnywhere Enterprise enables you to store values in parameters and variables to be used later for device data entry or to implement complex script logic such as loops and branches.

Parameters allow you to dynamically enter data into a device at the start of a script run, i.e., specify a different data value for each run. Parameter values are used as the basis for script logic, to specify device input, or to define a string of reference text. You can define parameters for actions or test cases. An action parameter, once created, can be used for all implementations. Test case parameters are not available for use in constituent actions.

Variables allow you to set/store values in your script to be used as the basis for script logic, to specify device input, or to define a string of reference text. Unlike parameters, variable values cannot be specified at the start of a script run. Variables can be *global* or *script specific*:

- *Script variables* are script specific, i.e., they are not available for use outside the test case or action in which they are created.
- *Global variables* are defined at the project level and can be used in any project script. The value stored in a global variable in one script can be used in another project script.

NOTE Variable and parameter names are case-sensitive. Be sure to reference variables and parameters exactly as named in DeviceAnywhere Studio.

3.5.1 Passing Parameters to a Visual Action

Use `setParameter()` to specify a parameter value in an `ActionHelper()` object, then `execute` your visual action, using `ActionHelper()` as an argument.

```
//// Passing a parameter and its value to a visual action

// 1. Create an ActionHelper object.
ActionHelper ah = new ActionHelper();

// 2. Set the primary device in the helper.
ah.setPrimaryDevice( device );
```

```
// 3. Set parameters and values for the action (e.g., myParameter = myValue).
ah.setParameter( "myParameter", "myValue" );

// 4. Call the visual action (myVisualAction, in this instance).
myVisualAction.instance.execute( ah );

// Java imports needed for the code sample above:
import com.mc.api.action.helper.ActionHelper
```

3.5.2 Using a Parameter in a Java Action or Test Case

This code example shows how to call a parameter that has already been defined (and has a value) from a Java action or test case. Be sure to call the correct parameter, i.e., a test case parameter from a Java test case and an action parameter from a Java action.

```
//// Using a parameter in a Java action or test case

/*
This sample Java action has two parameters:
stringParameter of type String
numberParameter of type Integer
*/

final String stringParameter = context.get("stringParameter");
final int numberParameter =
Integer.parseInt( context.get("numberParameter") );

/*
Use these parameters within this action and/or pass them to other actions
that take parameters.
*/
```

3.5.3 Using a Global Variable in a Java Action

This code example shows how to call a global variable that has already been defined (and has a value) from a Java action or test case. `getProject()` fetches the project context and `getValue()` gets the value of a variable.

```
////Calling a global variable

// Example 1. If your project variable is of type String:
final String myProjectStringValue =
this.getProject().getValue("myProjectVariableName");

// Example 2. If your project variable is of type Integer:
final int myProjectIntegerValue =
      Integer.parseInt( this.getProject().getValue("myProjectVariableName") )
;

// Use this value in your action.
```

3.5.4 Setting a Global Variable in a Java Action

This code example shows how to set the value of a global variable that has already been defined from a Java action or test case. `getProject()` fetches the project context and `getValue()` sets the value of a variable.

```
// Storing a local variable of type Integer or String in a project variable

// Example 1 - If your local variable is of type String:
String myLocalStringValue = "hello world";
this.getProject().setValue("myProjectVariableName", myLocalStringValue );

// Example 2 - If your local variable is of type Integer:
int myLocalIntValue = 123;
this.getProject().setValue("myProjectVariableName",
String.valueOf(myLocalIntValue));
```

3.6 Application Management

This section contains sample code to add applications to the Keynote DeviceAnywhere repository, upload them from there onto a device, and remove applications from a device.

3.6.1 Uploading an Application from the Repository onto the Device

Construct an `UploadApplicationHelper()` object, specifying the name, version, and type of the application to upload from the DeviceAnywhere repository onto a device. Then use the `uploadApplication()` function with `UploadApplicationHelper()` as an argument to load the application.

```
/*
Loading an application from the DeviceAnywhere repository onto a device
*/

// 1. Construct an UploadApplicationHelper object.
UploadApplicationHelper helper = new UploadApplicationHelper();

// Specify an application name in UploadApplicationHelper.
// Use exactly the name that is already in the repository.
helper.setApplicationName("YourAppName");

/*
2. Set the application type in UploadApplicationHelper according to mobile
platform.
*/
// Values are IPHONE, ANDROID_APK, BLACKBERRY_J2ME, J2ME, BREW.
helper.setApplicationType("IPHONE");

// 3. Specify application version in UploadApplicationHelper.
// Use exactly the same version that is already in the repository.
helper.setApplicationVersion("1.0");

// 4. Upload the application, using UploadApplicationHelper as an argument.
device.uploadApplication( helper );

// 5. Wait for 10 seconds to ensure that data has loaded correctly.
Thread.sleep(10000);

// 6. Except on iOS, reboot the device to complete the installation.
device.reset();

// Java imports needed for the code sample above:
import com.mc.api.device.helper.UploadApplicationHelper;
import com.mc.common.ApplicationType;
```

3.6.2 Removing All Applications

This sample code removes all applications (`removeAllApplications()`)from a selected device.

```
//// Removing applications

// 1. Remove all applications from a device.
device.removeAllApplications();


// 2. Reboot the device to complete application removal.
device.reset();
```

3.6.3 Adding and Uploading an Application

This code sample adds an application to the DeviceAnywhere repository and uploads it onto a selected device. Construct an `AddAndUploadApplicationHelper()` object, specifying the name, type, and location of the application to add to the repository and then upload onto the device. Then use the `addAndUploadApplication()` function with `AddAndUploadApplicationHelper()` as an argument.

```
//// Add and upload an application.

// 1. Construct an AddAndUploadApplicationHelper object.
final AddAndUploadApplicationHelper helper = new
AddAndUploadApplicationHelper();

// 2. Specify the application name in AddAndUploadApplicationHelper.
helper.setApplicationName("YourAppName" );

/*
3. Set the application type in AddAndUploadApplicationHelper according to
mobile platform.
Values are IPHONE, ANDROID_APK, BLACKBERRY_J2ME, J2ME, BREW.
*/
helper.setApplicationType( ApplicationType.IPHONE );

// 4. Specify the application file location in AddAndUploadApplicationHelper.
helper.setApplicationFile( new File("c:/Apps/AppName.app.zip"));

// 5. Add and then upload the application.
device.addAndUploadApplication( helper );

// 5. Wait for 10 seconds to ensure that data has loaded correctly.
Thread.sleep(10000);

// 6. Except on iOS, reboot the device to complete the installation.
device.reset();

// Java imports needed for the code sample above:
import com.mc.api.device.helper.AddAndUploadApplicationHelper;
import com.mc.common.ApplicationType;
```

3.7 Other Operations

Procedures covered in this section are specifying dependent JAR files in DeviceAnywhere Studio, waiting for a reference point defined in a state, collecting script execution return values and device screen proofs, and breaking execution when a step fails.

3.7.1 Adding External Dependencies to a Project

You can specify dependent JAR files in DeviceAnywhere Studio:

1 Right-click a project in the project list > **Properties**.

2 Select the **Dependencies** tab.

3 Click **Add JAR**.

4 **Select** a JAR file that your project is dependent on (you can only specify one JAR file at a time).

5 **Save** changes to project properties.

3.7.2 Waiting for a State

Instead of specifying a script-specific reference point (see Text Matching and Image Matching), you can call a reference point already defined in a state. A state is defined for all project devices. States consist of device-specific implementations to account for differences in interfaces. Use the `waitFor()` function to wait for a state.

```
//// Waiting for a state

// 1. Specify how long to wait for the state, in milliseconds.
final int timeout = 20000;

// 2. Wait for the state.
State s = myState.instance.waitFor(device, timeout);

// 3. See if the state was matched.
if ( s.getIsCurrentState(device) ) {
      // 3a. Device is in the specified state.
} else {
      // 3b. Device is not in the specified state.
}
```

3.7.3 Getting Return Values from Test Case Execution

This example demonstrates how to use the `isSuccess` and `isFailure` return values of the `ScriptReturn()` object for test case execution.

```
////Return values from a test case execution

// Execute a test case and get the ScriptResult object returned.
ScriptResult scriptResult = myTestCase.instance.execute(device);

// The actual return code is in the ScriptReturn object.
ScriptReturn scriptReturn = scriptResult.getReturn();

if ( scriptReturn.isSuccess() ) {
      // Script succeeded:
} else if ( scriptReturn.isFail() ) {
      // Script failed:
}
```

3.7.4 Getting Return Values from Action Execution

This example demonstrates how to use the `isSuccess` and `isFailure` return values of the `ScriptReturn()` object for action execution.

```
//// Return values from an action execution

// Execute an action and get the ScriptResult object returned.
ScriptResult scriptResult = myAction.instance.execute(device);

// The actual return code is in the ScriptReturn object.
ScriptReturn scriptReturn = scriptResult.getReturn();

if ( scriptReturn.isSuccess() ) {
      // Action succeeded:
} else if ( scriptReturn.isFail() ) {
      // Action failed:
}
```

3.7.5 Taking and Saving a Snapshot of the Current Device Screen

Use `getCurrentImage()` to capture a snapshot of the current device screen in supported formats.

```
//// Taking a snapshot of the current device and saving it to file

// 1. Take snapshot of the device screen.
BufferedImage image = device.getCurrentImage();

// 2. Create a File object to store the snapshot
// Formats available: JPEG, PNG, GIF, BMP and WBMP
File file = new File("image.bmp");

// 3. Write the image to the file.
ImageIO.write(image, "bmp", file);
```

3.7.6 Saving a Video of Test Case or Action Execution

You can capture a video of the device screen during script execution. To aid capture, create the `VideoCaptureParms()` object and specify video file format and frame rate.

Populate a `VideoHelper()` object with `VideoCaptureParms()` using the `populateVideoParams()` function. Call `captureVideoToFile()`, using `VideoHelper()` and the start and end frame numbers of your desired video as arguments.

```
//// Capturing video proofs

// 1. Get the start frame number.
int startFrameIndex = device.getCurrentFrameIndex();

// 2. Execute your Java action or testcase:
// TestcaseToBeRecorded.instance.execute(device);
// or ActionToBeRecorded.instance.execute(device);
// .....
// 3. Get the end frame number.
int endFrameIndex   = device.getCurrentFrameIndex();

// 4. Create VideoCaptureParams object.
VideoCaptureParams vcp = new VideoCaptureParams();

// 5. Set the frame rate (frames per second) in VideoCaptureParams.
vcp.setFrameRate(5);

// 6. Set the file format in VideoCaptureParams.
/*
Valid video types are:
FLV  - Flash
MPEG - MPEG 2
MOV  - Quicktime
AVI  - Uncompressed
*/
vcp.setVideoType("AVI");

// 7. Create a VideoHelper object and populate it with the Video Capture
Parameters
VideoHelper helper = new VideoHelper();
helper.populateVideoParams(vcp);

// 8. Save the zipped video to disk.
File videoFile = device.captureVideoToFile(startFrameIndex, endFrameIndex,
helper);

// 9. Specify the destination directory.
File dir = new File("c:\\destination-folder");

// 10. Move the file to the destination folder.
videoFile.renameTo(new File(dir, videoFile.getName()));
```

3.7.7 Breaking Execution

```
//// Abort script execution

/*
Action and test case failures are caught by DeviceAnywhere Studio during
execution. However, you can throw an exception to abort execution.
*/

/*
For example, if we are data driving our script, and the file containing
values is:
*/
File f = new File("data-input-file.txt");

// But the input data file is missing:
if ( !f.exists() ) {
      throw new Exception("input data file is missing!");
} else {
      // keep going
}
```