TestPlant Documentation

English 简体中文 日本語

收集和使用数据

在SUT上读取文本

ReadText和ReadTable使用eggPlant Functional的'OCR引擎，在SUT上指定区域之中，读取和返回文本。指定区域可以在矩形框内（由两个对角点确定），或在某点附近。

为简单起见，下文引用ReadText()函数，但所说明内容同样适用于ReadTable()。

读取矩形框内的文本

使用矩形框参数调用ReadText函数时，矩形框肯定会限制返回的文本值。也就是说，如果一行文本超出文本矩形框的边界，不会返回超出的文本 。

同eggPlant Functional内其它矩形框一样，文本矩形框也由两个对角点确定。每一点的信息可以由图像名称确定（图像名称代表查找到图像的位置），或者由其它坐标值确定。

示例：
```
Set address to ReadText (("AddressField", "EndAddressField"))
//将地址这一变量设置为指定矩形框内的文本。矩形框的对角点是AddressField和EndAddressField图像的位置。
```

读取某点附近的文本

使用某点参数调用ReadText函数时，OCR引擎会查找包含这一点的一行文本，或者查找在这一点附近开始的一行文本。

这一点的信息可以由图像名称确定（图像名称代表查找到图像的位置），或者由其它坐标值确定。

使用SUT剪贴板

如果所需读取的文本可以被选中复制，则可将文本复制到SUT剪贴板，然后使用RemoteClipboard()函数返回剪贴板内容。

示例：从SUT剪贴板返回文本

```
Click "SomeTextField"

TypeText CommandKey, "a"                    //选择文本字段内的所有内容。

TypeText CommandKey, "c"                    //将文本复制到剪贴板。

put RemoteClipboard(5)                      //等待5秒钟，返回之前远程命令生成的剪贴板内容；显示内容。
```

注：在Mac OS X系统上，不是以用户账户运行的VNC服务器（系统服务器），不能传输剪贴板内容。如果想要访问SUT剪贴板，确保使用有效的用户账号，连接到VNC服务器。

数据驱动测试

测试自动化的其中一个好处是，可以使用不同的数据值，重复运行测试。一个常用的方法是，将数据保存到文本文件，然后在脚本运行过程中，从文本文件读取值。SenseTalk内的直接文件存取和区块表达借助格式化的文本文件，使上述任务变得非常简单。

例如，要测试一个计算器应用程序，可以使用包含用逗号隔开的数字的文本文件。

示例：计算器测试数据文件

```
1, 2, 2

3, 4, n12

5, 6, 30
```

以下脚本指示计算器将文件内每一行的前两个值相乘。然后，返回结果并与第三个值比较验证。

示例：应用文本文件

```
repeat with theData = each line of file "CalculatorData.txt"

TypeText item 1 of theData                  //输入数据文件当前行的第1个值。

Click "Multiply Button"

TypeText item 2 of theData                  //输入数据文件当前行的第2个值。

Click “Equals Button"
```

示例：应用文本文件

```
Click “OutputField”

TypeText CommandKey, "a"                                                    //选择结果字段内的所有内容。

TypeText CommandKey, "c"                                                    //将文本复制到SUT剪贴板。

put remoteClipboard() into Answer                                           //将剪贴板内容添加到变量。

if (Answer is not equal to item 3 of theData) then                          //将结果同数据文件中的第3个值作比较。

LogError "Got:" & theAnswer & ", " & item 1 of theData & " x " & item 2 of theData & "should be "
& item 3 of theData                                                         //比较结果不一致则记录为错误。

end if

end repeat
```

创建数据文件

脚本中使用的数据可取自电子数据表和文本文件。电子数据表软件通常可将文件导出为CSV（逗号分隔值）格式，或者导出为制表符分隔的文本。

也可直接将数据添加到脚本，如下所示。

示例：创建数据文件

```
（该脚本创建每行有三个数字的测试数据文件。第三个数字为前两个数字的乘积。）

set file "/tmp/CalculatorData.txt" to {{      //在/tmp目录下创建一个CalculatorData.txt文件。（将该文件保存在可永久保存的位置。）

1,2,2

3,4,12

5,6,30

}}
```

注：SenseTalk利用“file+文件名”，可以轻松地读写文件所有内容。（参考上例中的文件名。）为确保使用正确的文件，最好使用文件的完整路径名。（更多信息，请参考*SenseTalk参考手册*中的“使用文件和文件系统”。）

读取和验证数据文件

以下是一个读取返回数据文件的脚本范例。

示例：读取数据

```
repeat with theData = each line of file "/tmp/CalculatorData.txt"           //依次将每行的内容分配到theData.txt。

put repeatIndex() & ":" & theData                                           //计算重复循环的次数；将该值写入每行首部。

end repeat
```

这时候运行该脚本，返回的Data.txt如下所示：

```
1:1,2,2
2:3,4,12
3:5,6,30
```

既然脚本可以读取数据文件，下一步就是验证每一行的值包含有用信息。

示例：验证数据

```
put zero into goodCount

put zero into badCount

repeat with theData = each line of file “/tmp/CalculatorData.txt”           // /tmp/文件的每一行...

put item 1 of theData into num1                                             //将每一行的三个数字添加到不同的变量。

put item 2 of theData into num2

put item 3 of theData into product

if product = num1 * num2 then                                               //比较第三个数字和前两个数字的乘积。

add 1 to GoodCount                                                          //如果相等的话，将增加GoodCount的值。

else
```

示例：验证数据

```
put "Bad Data at line " & repeatIndex() & ":" & theData                    //如果不相等，则显示“Bad data”消息。
add 1 to BadCount                                                          //并且增加BadCount的值
end if
end repeat
put "Good data lines:" & GoodCount                                         //返回GoodCount的值。
put "Bad data lines:" & BadCount                                           //返回BadCount 的值。
```

注：脚本中*line*和*item*这两个术语是SenseTalk的“区块表达”， 引用部分文本。通常，item由逗号分隔开，但是也可设置`itemDelimiter`全局属性，指定其它字符为分隔符。（区块表达和不同分隔符的使用的详细信息，请参考*SenseTalk参考手册*中的“区块表达”。）

对脚本事件计时

本节定义用于计时脚本事件的某些函数，并且举例说明如何在脚本记录中使用。（更多信息，请参考 *SenseTalk参考手册*中的“日期和时间函数功能教程”。）

- ***The Date***：返回当前日期。
- ***The Time***：返回当前时间。
- ***The Seconds***:返回从2001年1月1日开始的总共秒数。

示例：时间记录脚本

```
log "Starting timed task at" && the time && "on" && the date               //记录脚本开始运行的时间和日期。
put the time into startTime
(* 在此输入想要计时的代码。*)
put the time into stopTime
log "That took" && stopTime - startTime &&"seconds to complete."           //记录脚本运行的总时间。
```

输出：

```
2002-07-16 14:33:36 -0600log Starting timed task at 02:33 PM on 07/16/02
2002-07-16 14:33:39 -0600log That took 2.500326 seconds to complete.
```

‹ 重复使用代码 up 使用Omega13恢复错误 ›

Printer-friendly version