WinRunner 使用说明书

撰写：霍印鹏

审核：

文档编号：

目 录

1 引言

Winrunner 最主要的功能是自动重复执行某一固定的测试过程，它以脚本的形式记录下手工测试的一系列操作，在环境相同的情况下重放，检查其在相同的环境中有无异常的现象或与实际结果不符的地方。可以减少由于人为因素造成结果错误，同时也可以节省测试人员大量测试时间和精力来做别的事情。

在一次测试执行过程中，我们通过 WinRunner 记录所有必须的操作，设置检查点考察应用程序的各个对象，把过程保存为脚本。应用程序更新后，通过对脚本的运行，重复先前的操作，检查错误是否已被修改、是否有新错误被引入等。

WinRunner7.01 主要包括：GUI map、检查点、TSL 脚本编程、批量测试、数据驱动等几部分。参考资料为 WinRunner 在线帮助和 WinRunner 研究报告。

2 WinRunner 基本介绍

2．1 开始 WinRunner

单击 Start > Programs > WinRunner > WinRunner 来开始 WinRunner。注意 WinRunner 记录/运行进程的图标出现于视窗任务控制栏的状态区，这个进程建立并维护在 WinRunner 和被测的应用软件之间的连接。

开始运行WinRunner时，Welcome to WinRunner窗口将被打开。可以选择创建一个新的测试，打开一个现有的测试，或者看WinRunner的快速演示。

2．2 WinRunner 主界面

了解 WinRunner 的主界面，因为 WinRunner 的所有功能都是建立在该界面之中的只有了解了该界面才能更好的掌握 WinRunner。WinRunner 有如下要素 WinRunner 标题栏；WinRunner 命令菜单里的下拉式菜单栏；标准工具栏及经常在测试中使用的按钮；用户工具栏及经常在测试中使用的按钮；状态栏，有关现在的指令信息，插入点的行数和当前结果文件夹的名字。如图 2－1 所示

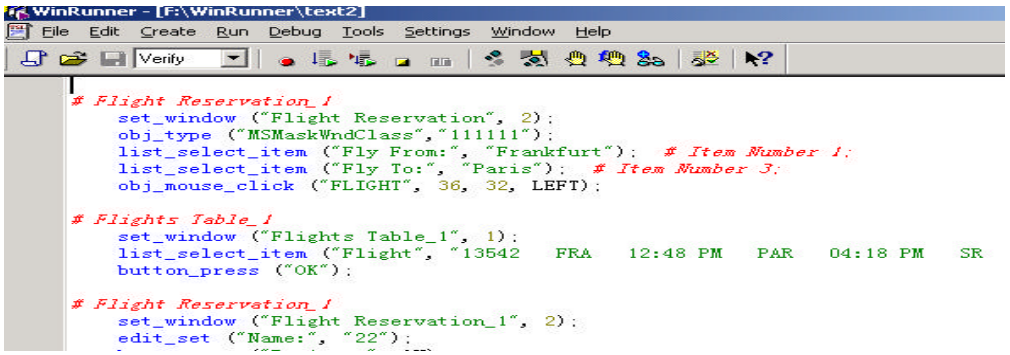


图 2－1WinRunner 的主界面

注意：在 7.01 版本中的主界面中在注释中书写标点符号的时候，要用半角的标点符号，不要用全角的标点符号。

3 两种录制模式

在进行录制脚本的时候，可以很快的录制出自动化操作的脚本。利用 WinRunner 可以记录测试人员的操作过程和在测试脚本语言中所体现的声明。在 WinRunner 中这些都是测试的脚本。其语言被称作 TSL（Test Script Language）的一种类 C 语言。在开始测试之前，测试人员要为测试中主要的地方或重要的阶段选择适当的记录模式，根据不同的情况，它在录制脚本的时候共包括两种测试的模式：

3．1 Context Sensitive（上下文相关）

该模式在测试中能记录测试人员绝大部分的操作过程。WinRunner 会确定测试人员每一个所点击的操作（包括窗口菜单、目录和按钮）和典型的操作任务。Context Sensitive 模式记录测试人员所选择的 GUI 对象（例如窗口，目录还有按钮等） 条件测试动作，虽然它忽略了关于对象在屏幕上的物理位置,但是每次录制测试脚本的时候,TSL 将综合描述 GUI 对象在执行时发生在测试脚本里的动作。WinRunner 对每一个选择的对象写一个唯一的 GUI 图来进行描述。GUI 图在于把文件维护从的测试脚本中分离。如果应用软件用户界面改变了，测试人员只要更新 GUI 图，不必重新录制。在测试之前，不仅仅运行过去的测试脚本还可以让测试人员对测试脚本进行编译和规划。WinRunner 模仿一个操作者移动鼠标指针、关闭应用软件、选择对象和加入键盘输入。WinRunner 在 GUI 图中读取与应用程序的 GUI 对象相匹配的逻辑名和物理描述，即使对象位置改变也能在窗口中定位。

在 GUI map 的“Greate”的菜单中点击“Record-Context Sensitive”就可以实现上下文相关的模式，一般在使用中默认为该模式。

3．2 Analog（模拟）

在该模式中 WinRunner 能精确的记录鼠标的点击和键盘的输入。例如：如果在注册窗口中点击“OK”按钮 WinRunner 能详细的记录了测试人员操作的动作。并且可以精确到测试人员所按鼠标的左按钮还是右按钮以及记录鼠标的移动轨迹。例如：画图的时候就可以记录鼠标所移动的轨迹。Analog 模式记录鼠标点击，键盘输入，并精确到鼠标行进的 X 和 Y 坐标值。一般在重要的程序或者录制画图功能的时候应用该模式。

在 GUI map 的“Greate”的菜单中点击“Record-Analog”后就可以实现模拟模式。

3．3 两种录制模式的区别

两种录制模式之间有以下区别：

上下文相关模式	模拟模式
它识别应用程序的 GUI 对象（用户界面）	应用程序包含位图的范围（和画图的范围）
不能记录精确的鼠标的动作	能够精确的记录鼠标的动作

4 相关测试

4．1 理解测试脚本

当点击对象 WinRunner 分配给该对象一个逻辑名，该逻辑名有助于测试人员对测试脚本的理解。例如：当选择命令，点击“ 飞行预定 ”(一个例子程序）的一个对话框，WinRunner 记录下列声明：

button_set("Order No."ON);

“ Order No。 ”是对象的逻辑名。

WinRunner 每次录制脚本的时候可以自动增加注释行。例如,当点击“ 飞行预定 ”的窗口的时候,WinRunner 产生下列注释行：

Flight Reservation

WinRunner 在每次开始工作的时候都会产生一个 set_window 的窗口。在窗口的内部产生下列的 set_window 的声明任务。当打开一个命令对话框 WinRunner 会产生下列声明：

set_window（“ aaa ”）；

edit_set(“ Edit ”,“ 3 ”)；

关于通过不同方式输入键盘值的 WinRunner 资料，可以察看 help->TSL Online Refernce。

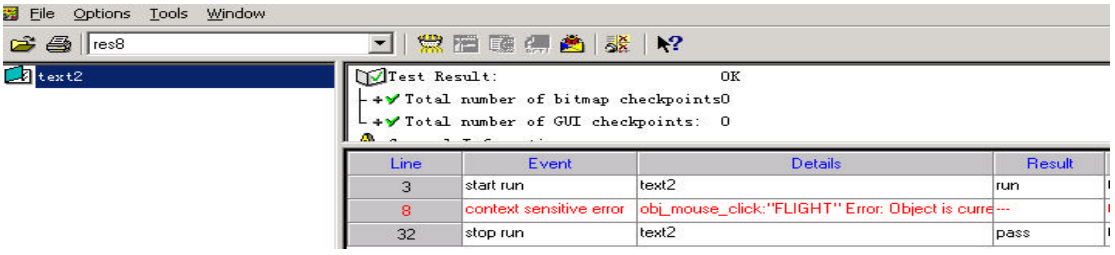
4．2 运行测试

运行所记录的测试脚本和分析运行结果的时候，WinRunner 提供三种运行模式，可以从工具栏中挑选择运行模式：

- 运行测试脚本，检测应用程序的运行情况，察看各检查点内容是否和预期一致，并保存测试结果，使用 Verify（校验）模式。该模式为默认运行模式。
- 检验测试脚本是否存在语法错误，使用 Debug（调试）模式。
- 更新已创建的用户界面检查点（GUI checkpoint）和位图检查点（bitmap checkpoint）的预期值，使用 Update（校正）模式。

4．3 分析测试结果

当一次测试运行结束以后，在 WinRunner 的测试结果窗口， WinRunner 将不同运行结果标以不同颜色（绿色的显示正确结束，红色的显示失败），测试人员能很快得知该测试是成功还是失败。也可以通过“ Tools ”菜单的子菜单“ Test Results… ”获得测试结果。如图 4－1 所示



如图 4－1 所示

如果在测试结果中有红色显示，可以双击该红色结果就会弹出错误的原因供测试人员参考。（注意：WinRunner 可以设定针对应用程序响应的等待时间。默认的等待时间是 10 秒，如果在这期间应用程序来不及响应，测试运行也可能是失败。这些可以通过延长等待时间或设置同步点予以解决。）

4．4 测试技巧

4.4.1 通配符

当应用软件的 GUI 的窗口或对象出现有规律的变化时可以用通配符

![内容].* （空格）

如： !Form.* ，表示为"Form"+任何字符串

!.* Form，表示为任何字符串+" Form"

4.4.2 文件路径

TSL 语言中，文件路径要用 " \\ " 表示 。

4.4.3 控件识别问题

有时录制脚本时发现有的控件识别不了（比如 WEB 页面中的标准控件识别不了），可能由以下情况造成：

1. 启动 WinRunner 时没有加载相应的 Add in(如启动时忘记加载了，可以通过 File>Test Properties…>Add in 来加载相应的插件)；
2. WinRunner 在应用程序之后启动；

4.4.4 自动加载 GUI Map 文件

```
static test_Path = getvar("testname"); # 得到测试用例的路径
static guifile_path = test_Path &" \\GUI_File.gui"； # GUI Map 文件名，建议将 GUI Map 文件保存在测试脚本的目录中，如果不是，将这里改为相应的绝对路径或相对路径
GUI_unload_all(); # 清空已经加载的 GUI Map
GUI_load(guifile_path); # 加载 GUI Map 文件
report_msg(test_Path); # 报告路径，调试脚本时用，正式执行可以不要
```

4.4.5 自动加载 GUI Map 文件

可以通过 toolbar_get_button_info function 函数来识别工具栏中某一按钮的状态。

4.4.6 字体专家(Fonts Expert）

在 Create>Get Text 中如果 WinRunner 无法识别窗口或对象中的字体、符号或数字，就需要用字体专家（Fonts Expert）来帮助识别字体：

首先要学习字体：

1. 在一个新的测试中打开 Tools>Fonts Expert
2. 选择 Fonts>Learn 打开学习字体的窗口
3. 在 Font Name 的文本框中输入一个名字
4. 点击 Select Font 按钮选择一个相应的字体
5. 点击 learn Font 按钮学习字体。当学习过程结束后，使其具有所识别窗口或对象中的字体或数字的特性。

创建一个字体群：

1. 选择 Font>Groups 打开字体群窗口
2. 在 Group Name 中选择或则新建一个 Group Name（如果是新建输入后应点击 New 按钮），然后从 fonts in Library 中选择相应的字体后关闭窗口。

应用字体群：

选择 Setting>General Options 选择 Text Recognition 的 Tab,选择 Use Image Text Recognition Mechanism 的检查框，在 Font Froup 中输入字体群，点击 OK 按钮。

4.4.7 键盘冲突

测试的过程中如果需要录制的应用软件的快捷键或热键和 WinRunner 的快捷键或热键有冲突，可以从以下两个途径来解决：

1. 如果是文本的快捷键如 Ctrl＋Z、Ctrl＋T 等可以通过 Settings>Editon Options…>Key assignments 的设置来更改相应的快捷键。
2. 如果是录制的热键如 Ctrl_L+F5 （Run from Top）、F6(step) 可以通过 Programs>WinRunner>Softkey Configuration 的设置来更改相应的热键。

4.4.8 选择录制对象

在测试过程中我们通过下面的设置可以有选择的录制脚本

1. 选择 Settings > General Options
2. 点击 Record tab.
3. 点击 Selective Recording .
4. 选择 Record only on selected applications .
5. 如果想录制开始菜单和浏览器选择 Record on Start menu and Windows Explorer .
6. 如果不像录制 Internet Explorer and/ or Netscape 取消 iexplore. exe and/ or netscape. exe.选项
7. 在列表中增加新的应用程序，点击一个空的位置，在对话框中输入应用程序的名字或使用浏览按钮选择应用程序
8. 如果只选择 Record only on selected applications 那么你仍然可以设置检查点和所有非录制的操作

4.4.9 录制 PowerBuilder 应用程序时应注意的问题

1. 在录制工具栏的时候可能会出现录制脚本重复的现象通过下面的设置步骤可以解决这个问题：

1) Start WR

2) Open a test at <WR installed folder>\lib\pbinit

3) Insert the following statements at the very end of the test

```
set_class_map("FNFIXEDBAR60", "object");
set_record_attr("FNFIXEDBAR60", "class", "label MSW_class attached_text ","location");
add_cust_record_class("FNFIXEDBAR60", dll_path,"PBToolBarHLRec", "", "", "", "");
```

4) Close WR and restart WR with PowerBuilder Add-in selected.

在运行脚本时录制的 PowerBuilder 的工具栏可能不像录制的菜单那么可靠。建议在录制脚本时使用菜单代替工具栏。

2. 当用 WinRunner 录制表格中新增记录的时候，会产生 tbl_get_selected_cell()的 TSL 的脚本，但是当脚本回放的时候由于新增行，行号变了脚本就无法识别对象进行回放。通过下面的操作可以解决这个问题。

   tbl_get_rows_count(table,rows_count); #这个函数获取最近的行号

   tbl_get_selected_cell (table, out_row, out_column);

   通过 tbl_get_rows_count()来获取起行号，把这个值传递到 tbl_get_selected_cell()的“out_count”中来。

   注意：在 tbl_get_selectd_cell（）中有时可能不能直接用 rows_count 的值因为表的行中需要用到“#”所有的行号都要写在“#”后。

4.4.10 其它

下面是在使用 WinRunner 的时候为防止出现意外，需要注意的一些问题：

- 在开始之前关闭不必要测试的软件。
- 知道创建测试在哪里开始和在哪里结束。
- 在模拟模式中要避免不必要的重复按鼠标。
- 当从上下文相关模式转化到模拟模式时，必须先使用鼠标激活应用程序窗口。
- 在运行测试中，有时候应用程序的反应时间可能会长一些。
- 在使用检查点的插入语句的时候要注意光标在测试脚本中停留的位置。
- 使用 Bitmap Checkpoint 来检查位图的时候需要注意测试机屏幕的尺寸、颜色和屏幕区域的设置，否则有可能会影响检查的结果的正确性。

5 检查点的应用

设定检查点可以检查所设定区域的显示是否和预期结果相符。通过检查点的设置以及对各点处输出信息的编程定义，我们可以在脚本运行结果单中查看各项测试内容是否都已通过。在功能测试中，检查点可以用在以下两个方面：检查应用程序经过修改后对象状态是否发生变化；检查对象数据是否和预期数据一致。WinRunner 提供的检查点类型有：GUI checkpoint、Bitmap checkpoint、text checkpoint 等。

5．1 用户界面检查点（GUI checkpoint）

在程序运行过程中，当输入条件发生变化时，一些相应 GUI 对象的状态也会发生变化，如编辑框的内容、单选按钮被选中、菜单项或按钮的禁用等，这些都是靠开发人员编程实现的。但在软件交付开发人员修改时，对某一模块的改动可能会引起另一模块错误的产生。针对这种情况，在需要考察的对象上设置检查点，软件改正后重新测试时，只要运行先前测试中录制的脚本，就可以发现是否有上述情况发生。

对同一对象，我们可以考察它的多个属性，这在脚本录制过程中设置 GUI checkpoint 时相应设定。脚本回放后，结果列表里就可以显示对象各个属性值的变化情况。

选择 Update 脚本运行模式，可以更新已保存的 GUI 对象状态。

5．2 位图检查点（bitmap checkpoint）

应用程序可能包含位图区，比方说图形或图表，设置 bitmap checkpoint 就能够以象素为单位逐一对前后两个版本中的位图进行比较，得出是否一致的结果。

举例而言，在一个应用程序中，点击“ 清除 ”按钮后应该清空某一位图区域。我们录制脚本时，在点击按钮这个动作之后对此区域设置 bitmap checkpoint（checkpoint 记录下区域为空白状态），将来利用这一脚本测试新版本的应用程序时，就可以检查“ 清除 ”按钮的功能是否依旧有效。如果位图区域无法被清空，检查结果中就会报错。

Checkpoint 的范围确定有两种方式：以窗口为参考对象或以屏幕为参考对象。

5．3 文本检查点（text checkpoint）

位图或 GUI 对象里一般都包括一些文本信息，如字符串、数字等，text checkpoint 可以从应用程序中读取这些信息，并验证信息的正确与否。

检查的过程如下所示：

1．设置 checkpoint，脚本录制时 WinRunner 自动从中取数据；

2．预先计算数据的正确变化情况，以此为据对脚本编程；

3．对新版本应用程序运行此脚本，从结果报告中观察数据变化是否与预期相符。

这里有一段 TSL 代码：

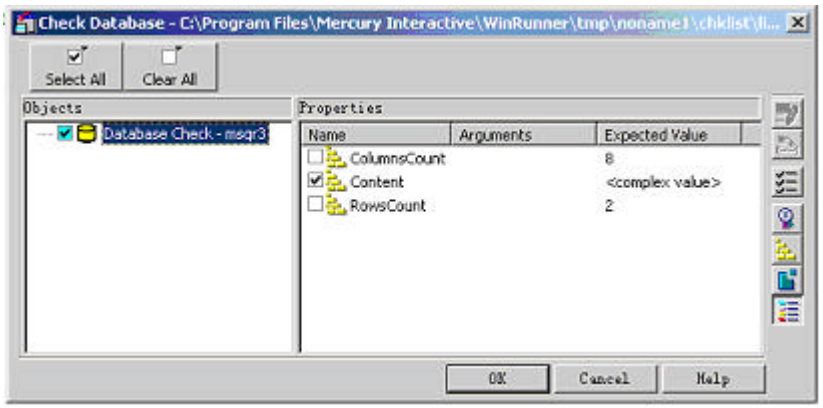
```
if (new_ total = = first_ total + 1)    //new_total和first_total都是检查点的数据
      tl_ step ("graph total", 0, "Total is correct.");    //数据变化正确时的显示
else
      tl_ step (" graph total", 1, "Total is incorrect.");    //数据变化错误时的显示
```

GUI checkpoint 和 text checkpoint 的区别：读取标准 GUI 对象（如编辑框、列表、菜单等）中的信息，使用 GUI checkpoint 读取位图或非标准 GUI 对象的信息，使用 text checkpoint。

5．4 数据库检查点（database checkpoint）

通过设置 database checkpoint,可以对 ODBC 数据源提供的数据库表的一些诸如表行列数、主键、数据内容作检查。可以通过以下的几步来实现：

在 Greate 下的 Database check point 中选择 database custom check，按照提示，选择要检查的数据源的连接方式（现有的只支持 ODBC 数据源）和所用的查询语句的来源，这里所指定的查询语句用来确定所设置的检查点涉及到的数据范围，查询语句的来源可以是指定一个已有的.sql 文件中，也可以是语句的拷贝，或者新建一个，查询之后，数据库表中的情况也是按对象和对象属性的形式描述的，可以对这些属性作一些限定，形成对这个检查的预期（如图）。



这样就在脚本中插入了一段 TSL 的代码，例如：

db_record_check("list1.cvr", DVR_ONE_OR_MORE_MATCH, record_num);

在 Greate 下的 Database check point 中还有一种检查点 runtime record check，用来将运行测试中采集到的值与数据库表中的数据作比较，然后在测试结果中列出比较的结果。在脚本中这种比较的 TSL 如下：

db_check("list2.cdl", "dbvf3");

5．5 同步检查点（Synchronization Point）

这种检查点的设置过程和普通的检查点的设置基本一致，用来在规定的时间内等待预定的内容出现，包括对象的属性的值、对象位图和区域位图。一般会插入如下语句之一：

obj_wait_info("RichEdit20W","enabled",1,10);（对象的属性的值）

obj_wait_bitmap("RichEdit20W", "Img1", 1);（对象位图）

win_wait_bitmap（"lb_WinRunner.doc","Img4", 1, 85, 203, 79, 75）；（区域位图）

6 脚本编程

6．1 插入函数

在用 TSL 编写测试用例时，首先要通过录制产生一些基本的测试脚本，再用 Function Generator 插入函数。

插入函数的目的是为了测试一些特定的功能。这是用 TSL 编写测试用例的基本要求。在界面“ Insert Function ”来实现其功能

6．2 过程中局部逻辑正误的判断

通过 Debug 我们可以既容易又快速地检查出例程的语法错误和逻辑错误。我们可以一步一步地运行测试用例，可以在脚本中设置断点，使测试用例停止在特殊的地方，也可以利用观测器观察变量和表达式的变化。我们在调试测试脚本时应该运行在 Debug 模式下。

6．3 WinRunner 的基本命令

我们可以在菜单栏选择WinRunner命令。另外，某些WinRunner指令能通过按软键实行。

6.3.1 从菜单选择命令

可以从菜单栏上选择所有的WinRunner命令。

6.3.2 工具栏上的单击命令

可以在工具栏上单击鼠标执行一些WinRunner命令。WinRunner工具栏中有两个built-：标准工具栏和常用工具栏。测试人员可以制定一些使用最频繁的命令在用户工具栏上。

6.3.2.1 创建移动的工具栏

可以把一个工具栏变成移动的工具栏。使测试人员能够用移动的工具栏的命令将WinRunner减少到最小以维持存取，所以测试人员能自由的对被测试的软件操作。

6.3.2.2 标准工具栏

标准工具栏包括在运行测试习惯的命令按钮。它同样包括打开和存储测试脚本按钮，显示测试报告按钮，和访问帮助按钮。

6.3.2.3 常用工具栏

常用工具栏包括创建测试时常使用的按钮。在没有特别指定的情况下，用户工具栏被隐藏起来。要显示用户工具栏，在窗口菜单上选择它。当它显示的时候，它的默认位置是WinRunner窗口右边缘。

用户工具栏时可制定的工具栏。可以添加或移动按钮在被测试的应用软件中经常用的普通按钮。下列按钮默认情况下不出现于用户工具栏中：

Record - Context Sensitive	（纪录—上下文敏感的模式）
Stop	（停止）
GUI Checkpoint for Object/ Window	（对象/窗口的GUI检查点）
GUI Checkpoint for Multiple Objects	（多数的对象GUI检查点）
Bitmap Checkpoint for Object/ Window	（对象/窗口的位图检查点）

Bitmap Checkpoint for Screen Area （显示区域的位图检查点）
Database Checkpoint （数据库的检查点）
Synchronization Point for Object/ Window Property（对象/窗口的同步点检查）
Synchronization Point for Object/ Window Bitmap（对象/窗口位图的同步点检查）
Synchronization Point for Screen Area Bitmap （显示区域的同步点检查）
Get Text from Object/ Window （从对象/窗口得到测试）
Get Text from Screen Area （从显示区域得到测试）
Insert Function for Object/ Window （为对象/窗窗口插入功能）
Insert Function from Function Generator （来自功能发生器的插入功能）

7 数据驱动测试的创建(一个脚本用于多组数据)

当测试顺利的调试和运行测试后，测试人员可能想为相同的测试任务配置多组数据，以检验运行结果。在做这些的时候要应用数据驱动方式，创建相应的驱动数据资料表格安置数据。

数据驱动测试必须包括下列几步：

- 增加用于打开和关闭数据表格的脚本语句；
- 增加脚本语句和函数，从数据表格中读取数据，并循环运行，以便每一组数据都能被执行到；
- 将录制的语句以及检查点语句中的固定值用参数替换，这被称之为测试参数化。

测试人员可以通过 DataDriver Wizard 将测试脚本转变为数据驱动测试，或者直接手动修改测试脚本。

8 批量测试（连续运行多个测试脚本）

8．1 创建、运行批量测试

- 包含 CALL 声明，打开另一个测试，例如：
  call “ C:\\qa\\flights\\lesson9 “();
  运行一个测试的期间,WinRunner 认为这是一个调用声明，并运行这个调用测试。当调用的测试完成以后，WinRunner 又返回原来的测试继续运行。
- 选择标有 Run in batch mode 的选项的对话框（Setting>General Options）在测试运行之前。这个选项在测试中禁止运行中断。例如，如果位图有错误他就会提示但不停止运行测试。

在回顾这批运行测试结果的时候，可以看见它们运行的全部结果（通过或失败），所显示的都是可用的结果。

8．2 分析这批测试结果

一旦运行结束就可以分析 WinRunner 测试窗口的运行结果了。批量测试的结果全部在窗口展示（通过或失败），每个正确的结果会返回显示测试成功。错误的结果会返回显示测

试失败。

9 GUI 对象

9．1 两种 GUI 模式

WinRunner 的 GUI 共有两种模式：GUI Map File Per Test mode 和 Global GUI Map File mode。可以通过 Settings>General Options>Environment 页面中的 GUI Map File Mode 选项对其进行设置。

9.1.1 单一 GUI 模式（GUI Map File per Mode）

GUI Map File Per Test mode 每次录制脚本的时候能够自动创建一个新的 GUI 文件，且其 GUI 文件无法更改，每次运行测试的时候可以自动加载。这种模式通常适用于初学者或者被测软件的 GUI Map 确定不会改变的情况。

9.1.2 全局 GUI 模式（Global GUI Map File Mode）

Global GUI Map File Mode 在这个模式中可以使用一个 GUI Map 应用在一批测试脚本中，也可以根据不同的对象对 GUI Map 进行修改。其 GUI 默认保存在临时目录中，需要在保存脚本的同时保存 GUI Map 信息。

9.1.3 两种模式的区别

两种模式	单一模式（GUI Map File per Test）	全局模式（Global GUI Map File）
方法	在测试的过程中将自动保存 GUI 信息，打开测试时可以自动加载 GUI 文件	在测试的过程中需要保存 GUI，当应用程序改变时必须更新 GUI 文件
优点	1. 每个测试都有自己的 GUI 文件 2. 在关闭程序的时不必保存 GUI 3. 维护和修改简单	1. 当对象或窗体的描述改变，对应的 GUI 文件的属性也得作相应的修改 2. 维护或修改效率高
缺点	只要应用程序的 GUI 改变都必须更新 GUI 文件	当新建 GUI 或运行测试脚本时必须保存或装载 GUI 文件
建议	适用于初学者或被测软件的 GUI 不会产生变化	适用于经验丰富的 WinRunner 使用者，或被测软件的 GUI 可能会产生变化

Edit GUI Map 时的注意事项：

1. 一个单独的 GUI 文件中不能包含两个相同逻辑名的窗口。
2. 一个单独的 GUI 在同一窗口中不能有两个相同逻辑名的对象。

Merge GUI Map 时的注意事项:

1. 两个窗口带有相同的逻辑名，但其物理描述不同。

2. 相同的窗口中的两个对象带有相同的逻辑名，但是其物理描述不同。

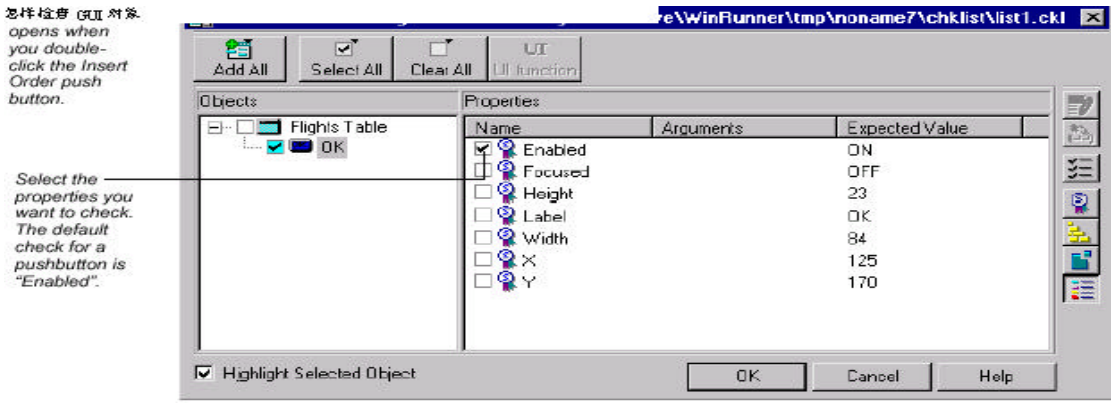
9．2 检查 GUI 对象

在利用 WinRunner 测试的过程中,GUI 是测试的主要的对象，只有充分的掌握了 GUI 才能更好的进行测试。

创建 GUI 检查点对 GUI 对象进行检查，一个 GUI 检查点可以检查一个属性对象的行为，.例如检查：扫描内容、按钮是否激活、是否是焦点、按钮的高度和宽度等。

9.2.1 创建测试

创建一个单独的 GUI 对象检查点（single object），首先指向要检查应用软件的对象。如果单击对象，就会弹出一个带着缺省选项的校验表并插入到测试脚本中。一个校验表包含了关于 GUI 对象的信息和各对象可被检查的属性。如果双击对象，这个检查 GUI 对话框就会打开和展示可挑选检查的属性。挑选想要检查的属性后点击 OK 就可把校验语句插入到测试脚本中。如下图所示



如图 9－1

无论是选择检查一个默认的属性还是指明其他需要考察的属性，WinRunner 都将捕获属性的当前值并保存为测试中的预期值。随后 WinRunner 将一条 obj_check_gui 语句插入到测试脚本中，用于检查对象的属性；或将一条 win_check_gui 语句插入到测试脚本中，用于检查窗口属性。

在一个新的版本的软件中运行这个测试，WinRunner 将对象的期待行为和应用程序的实际行为进行比较，并提供比较结果。

9.2.2 增加 GUI 检查点的测试脚本

这里通过例子说明如何增加的测试脚本

打开 WinRunner 的时候，检查飞行程序 Open 命令对话框的一些功能。

1.Start WinRunner and open a new test

开始 WinRunner 并打开一个新的测试

如果 WinRunner 没有打开，选择 Programs>WinRunner>WinRunner 在 Start 菜单。如果在欢迎窗口打开。点击 New Test 按钮。另外选择 File>New.一个新的测试窗口打开了。

2.开始运行飞行预约程序

选择 Programs>WinRunner>Sample Applications>Filght 1A 在 Start 菜单，在注册窗口。填写名字和密码。点击 OK，重新配置飞行保留程序和 WinRunner。

3.在 Context Sensitive mode 模式中开始录制

开始记录上下文敏感的模式

选择 Create>Record-Context Sensitive 或则点击 Record 按钮

4.打开对话框的 Open 命令

选择 File>Open Order 在飞行程序中如图所示

图 9－2

5.给对话框创建一个 GUI 检查点

选择 Create>GUI Checkpoint>For Object/Window 或者点击 GUI Checkpoint for Object/Window 按钮。使用手指指示器单击 Order No.检查框。这个用户界面检查对话框打开和展开一个有用的检查。注意：如果单击对象则这个对话框不打开（要想是对话框打开必须双击对象）。同意默认的检查，“ 声明 ”这个检查捕获当前的声明（off）和点击对话框保存期待的结果。

在检查用户界面对话框中点击 OK 插入到检查点的测试脚本中。这个检查点作为一个 obj_check_gui 声明

6.输入 4 作为命令

选择 ORder No.检查对话框对应的文件框中输入 4

7. 为 Order No.创建另一个用户检查点检查对话框

选择 Create>GUI Checkpoint>for Object/Window 或点击 GUI Checkpoint for Object/Window 按钮使用手指指示器单击 Order No.检查框。WinRunner 立即插入检查点在测试脚本（一个 obj_check_gui 声明）这个检查是默认的检查“ State ”这个检查捕获当前的声明（on）检查框作为保存的结果。

8.创建一个消费者的名字的检查点

选择 Create>GUI Checkpoint>For Object/Window 或点击 GUI Checkpoint for Object/Window 按钮。使用手指指示器双击 Customer Name 检查框。检查用户界面对话框打开和展示可用到的检查。同意默认的检查“ State ”和挑选“ Enabled ”作为另外的检查. 这个规定的检查捕获当前的声明（off）的检查框；激活检查捕获当前的条件（off）的检查框。点击 OK 在检查 GUI 对话框中插入检查点在测试脚本中。这个检查点出现作为一个 obj_check_gui 声明。

9.点击 OK 打开命令对话框直到打开命令

10.停止录制

选择 Create>Stop Recording 点击 Stop 按钮

11.保存测试

12.如果 GUI Map 使用的是全局 GUI 模式，则在退出 WinRunner 时需要保存 GUI。

选择 Tools>GUI Map Editor.选择 View>GUI Files.选择 File>Save 点击 Yes 增加 GUI 图形。选择 File>Exit 关闭 GUI 图形的编辑。用户界面图形出现作为 obj_check_gui 或

win_check_gui 声明在测试脚本中。例如：

Obj_check_gui("Order No.","list1.ck1","guil",1)

Order No.**对象的逻辑名字**

list1.ck1 **是选择的较验表（包括设置的检查）**

gui1 **是文件包含捕获的 GUI 的数据**

1 **是时间（在第二秒中）需要完成检查。这个时间增加了确切的含义在** Timeout_msec **的测试选项。**

9.2.3 运行测试

1.**打开一个飞行预约请求的应用程序**

2.**在 WinRunner 里，在使用者工具栏中选择较验模式**

3.**选择从顶部运行**

选择 Ren>Run from Top，或点击 Run from Top 按钮，这个 Run Test 对话框打开了。接受默认的名字“ res1. ” 在检查对话框中挑选 Display Test Results at End of Run.

4.**运行测试**

在测试对话框中点击 OK

5.**回顾测试结果**

当运行测试完全结束以后，在 WinRunner 的窗口中会显示测试结果。在测试日志的结果部分“ 结束 GUI 检查点 ”事件中如果出现绿色字体说明测试成功，如果是红色字体说明测试失败，要分析测试结果。

双击已经结束的用户界面图形的检查点。GUI 检查的结果弹出对话框。选择 Customer Name 到展示对话框的列表。如图所示

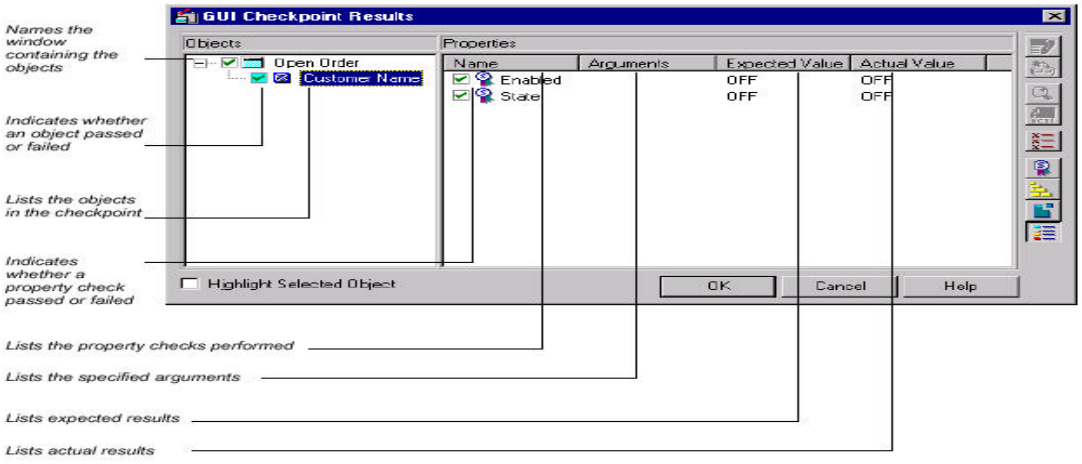


图 9-3

6.**关闭测试结果。**

点击 OK 到关闭 GUI 检查点结果对话框。然后选择 File>Exit 到关闭测试结果窗口

7.**关闭飞行预约程序**

选择 File>Exit

9.2.4 运行一个新的版本的测试

在这个练习将要运行测试一个新的版本的飞行预约程序并注意 GUI 对象的检查点

1.Open version 1B of the Flight Reservation application

在开始菜单中选择 Programs>WinRunner>Sample Applications>Filght 1B。

2.制作一个可以正常运行的测试

在 WinRunner 中选择上面的脚本。

3.在工具栏中选择较验模式

4.选择从顶部运行

选择 Run>Run from Top.或点击 Run from Top 按钮。运行测试对话框打开了。接受默认的名字。在检查对话框中挑选 Display Test Results at End of Run.

5.运行测试

点击 OK。运行开始。

6.回顾测试结果 。

当测试完全结束以后。WinRunner 的测试结果就会显示出来。在测试的航行日志部分。一个“结束 GUI 检查点”，在结果中红色的代表了测试失败。这个结果是通过一个或更多的测试检查出来的。双击红色的“结束 GUI 检查点”可以详细的介绍失败的原因。这个 GUI 检查点结果对话框的打开 。消费者的名字的对话的错误如图所示

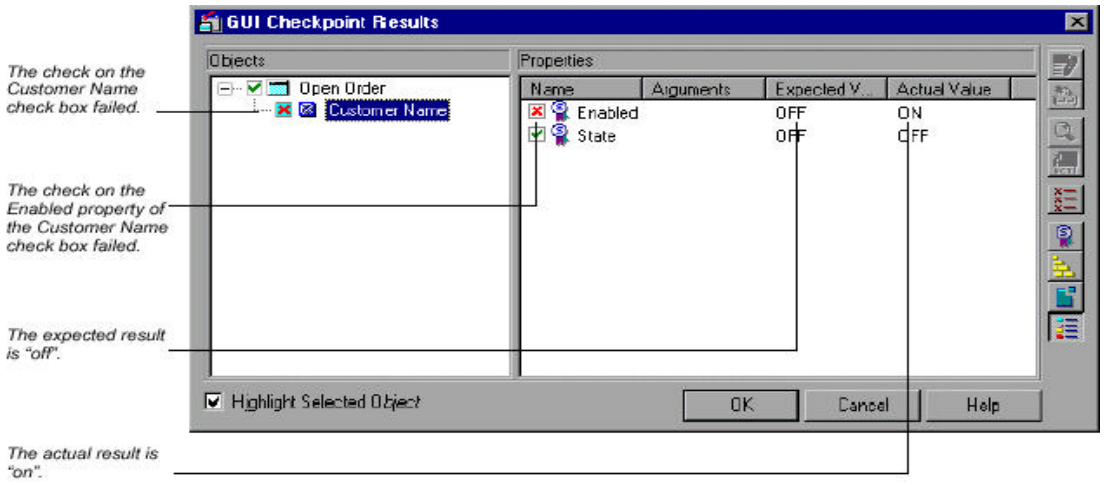


图 9-4

7.关闭测试结果窗口

点击 OK 在 GUI 检查点结果对话框和选择 File>Exit 关闭的测试结果窗口

8.关闭测试

选择 File>Close

9.关闭译文 1B 飞行预约程序

选择 File>Exit

9.2.5 运行 GUI 检查点的技巧（GUI Checkpoint Tips）

- 在测试检查中创建几个的 GUI 检查点。选择 Create>GUI Checkpoint>For Multiple>Objects 后，创建 GUI 检查点的对话框打开。选择几个想要完成详细检查的 GUI 对象的检查点。创建检查点后，WinRunner 插入一条 win_check_gui 语句到的测试

脚本中来，语句中包含有该被选对象的检查点列表。

- 让 WinRunner 在出现错误的时候不提示错误的信息而一直运行，使其可以运行一个通宵的测试甚至更多。选择 Settings>General Options，在选项对话框中，点击 Run 属性页，清除 Break when verification fails 单选框，可以使测试运行不带停顿。
- 关于如何设置与测试脚本运行有关的选项，可以察看 WinRunner User's Guide 的“Setting Global Testing Options”和“Setting Testing Options from a Test Script”两章。如图所示

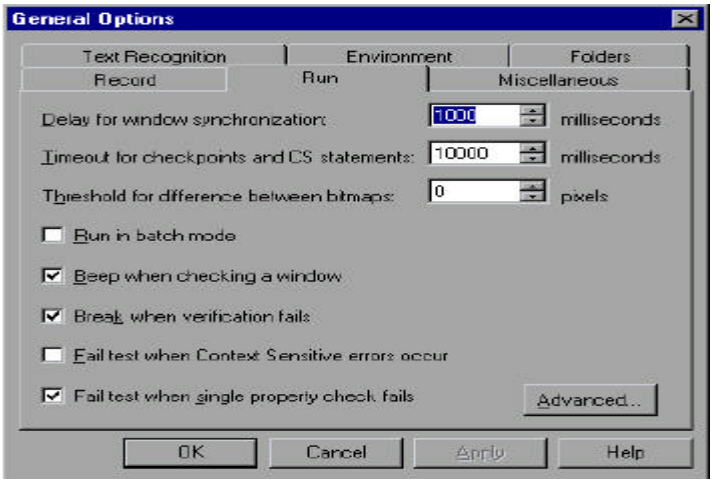


图 9-5

- 如果某个 GUI 检查点的预期值有了变化，需要在校正模式中运行测试。WinRunner 在运行中捕获新的值，更新原有的 GUI 预期数据。
- 关于 GUI 检查点的更多信息，可以查看 WinRunner User's Guide 的“Checking GUI Objects”一章。

9．3 检查位图

9.3.1 如何检查位图

如果应用程序包含位图，例如制图或者曲线图，我们可以利用位图检查点（bitmap checkpoint）检查这些涉及到的范围。位图检查点通过逐一对比各个像素比较捕获的位图。

创建位图检查点，必须根据需要确定一个范围，窗口或对象。例如：如图所示

图 9-6

WinRunner 捕获位图图像作为期待的结果。每当捕获到一个对象，WinRunner 插入一条 obj_check_bitmap 语句到测试脚本中，或者通过 win_check_bitmap 语句检查一个窗口或一个范围。

当测试脚本运行于新版本的应用程序时，WinRunner 比较已保存的位图和应用程序中的实际位图，如果发现有差异，能从测试结果窗口中显示有关区别内容的图形。

9.3.2 增加位图检查点的测试脚本

这个练习中将要在传真命令的对话框中测试代理签名。要使用位图检查点检查那个标记名字的对话框。使用另一个位图检查点来检查当点击清除按钮的时候该对话框是否清除。

1.用 WinRunner 打开一个新的测试

如果 WinRunner 没有打开，在开始菜单中选择 Programs>WinRunner>WinRunner。如果欢迎菜单已经打开。点击 New Test 按钮。否则，选择 File>New. 一个新的测试窗口打开了。

2.开始飞行预约程序并注册。

在开始菜单中选择 Programs>WinRunner>Sample Applications>Filght 1A。

3.开始录制上下文敏感的模式

选择 Create>Record-context Sensitive 或点击 Record 按钮在工具栏

4.打开命令#6

在飞行预约窗口，选择 File>Open Order.

在飞行预约程序中，在打开命令的对话框中选择 File>Open Order 选项，在 Order No. 的文本框中输入“6”，点击 OK 来打开命令。

5.打开传真命令对话框

选择 File>Fax Order

6. 在传真号对话框中输入一个 10 位的阿拉伯传真数字。

7.选择 Send Signature with Order 选项

8.转换类似体模式

按 F2 或在键盘点击 Record 转换模式

9.在代理签名的对话框中标记你的名字

10.转换到上下文敏感模式

按 F2 或键盘点击 Record 按钮转换上下文敏感模式

11.插入一个位图检查点检查签名

选择 Create>Bitmap Checkpoint>For Object/Window 或点击 Bitmap>Checkpoint for Object/Window 按钮。

使用手指指示器点击 Agent Signature 对话框。WinRunner 捕获位图和插入一个 obj_check_bitmap 声明在测试脚本中。

12.点击清除署名按钮

这个署名从 Agent Signature 对话框中清除

13.插入另一个位图检查点检查署名对话框的代理商。

选择 Create>Bitmap Checkpoint>For Object/Window 或点击 Bitmap Checkpoint for Object/Window 按钮

使用手指指示器点击 Agent Signature 对话框。WinRunner 捕获一个位图和插入一个 obj_check_bitmap 声明在脚本中。

14.点击取消按钮在传真命令对话框中。

15.停止录制

16.保存测试

17.如果工作在全局 GUI 模式中，保存一个新的 GUI Map 对象。

选择 Tools>GUI Map Editor.选择 View>GUI Files.选择 File>Save.点击 Yes 或 No 来增加新的对象或性的窗口到 GUI 图形中来。选择 File>Exit 关闭 GUI 图形编辑。

如果想更多的了解保存新的窗口和保存新的界面的信息。

位图检查点发表作为 obj_check_bitmap 或 win_check_bitmap 声明在测试脚本中,例如：

obj_check_bitmap("(static)","Img1",1);

static 对象范围的逻辑名字

Img1 是文件包含捕获的位图。

1 是时间（在秒）必须来完成的检查。这个时间是增加 timeout_msec 测试选项的价值。

9.3.3 显示期待的结果

1.打开 WinRunner 的测试结果窗口

选择 Tools>Test Results 或点击 Test Results 按钮。这个测试结果的窗口被打开。

2.显示捕获的位图

在测试的注册部分。首先双击"capture bitmap"事件，或选择和点击 Display 按钮如图所示。

如图 9-7

下一步，双击“ capture bitmap ”事件，或选择它和点击 Display 按钮

3.关闭测试结果窗口

关闭位图和选择 File>Exit 关闭测试结果窗口。

9.3.4 运行一个新的测试版本

在一个新的版本中运行飞行预约程序中运行行测试。

1.关闭飞行预约测试 1A

选择 file>Exit

2.开始运行飞行预约测试 1B

在开始菜单中选择 Programs>WinRunner>Sample Applications>Flight 1B。

3.制作可正常运行的测试

选择上面的测试脚本。

4.检查较验（Verify）模式挑选工具栏

5.选择从顶部运行

选择 Run>Run from Top,或点击 Run from Top 按钮，这个 Run test 对话框被打开。接受默认的测试的名字。选择 Diaplay test results at end of run 挑选检查框。

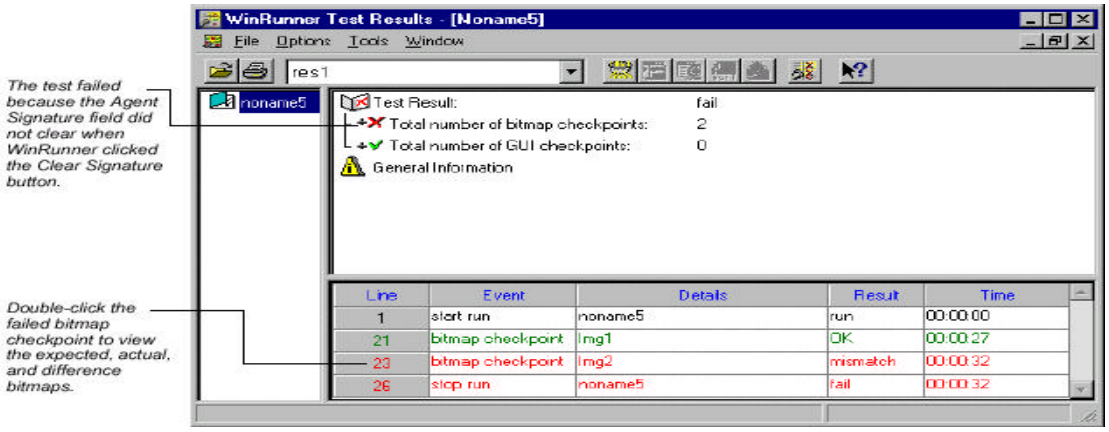
6.运行测试

点击 OK 测试开始

如果位图检查点出现错误，点击 Continue 信息窗口。

7.回顾测试结果

当测试结果结束。这个测试结果出现在 WinRunnner 的测试结果的窗口中。如图所示



如图 9-8

8.关闭窗口的测试结果

选择 File>Exit 关闭测试结果窗口

9 选择上面的测试脚本

选择 File>Close

10.选择校验（Version）1B 的飞行预约程序

选择 File>Exit

9.3.3 检查位图的技巧

- 为了获取一个范围，选择 Create>Bitmap Checkpoint>For Screen Area 或点击 Bitmap CheckPoint for Screen Area 按钮在使用者工具栏中看菜单按钮（此时一个测试脚本必须已经打开，否则菜单项或按钮都是不可选的）。使用十字鼠标指针在指定的窗口中划定一个范围。

WinRunner 在测试脚本中插入一条 win_check_bitma 语句。该语句带的参数定义了范围所处的位置（x-和 y-坐标）和尺寸（宽和高）。

- 让 WinRunner 在出现错误的时候不提示错误的信息而一直运行，使其可以运行一个通宵的测试甚至更多。选择 Settings>General Options，在选项对话框中，点击 Run 属性页，清除 Break when verification fails 单选框，可以使测试运行不带停顿。
- 当运行测试包含位图检查的时候，如果屏幕设置（如分辨率）不同，WinRunner 将要报告位图匹配不当。
- 如果要更新位图检查点的预期结果，在校正（Update）模式中运行测试，WinRunner 将自动更新现有的预期结果。

关于位图检查更多的信息，可以查看 WinRunner User's Guide 的“ Checking Bitmaps ”一章。

10 TSL 测试脚本语言

10．1 使用 TSL 语言对脚本编程

当录制一个脚本的时候，每次点击 GUI 对象或用键盘输入时，在 WinRunner 的测试脚本中产生了一个 TSL 的声明。可以增加录制 TSL 的函数，TSL 包括许多构造函数能增加脚本测试函数的功能和适应性。在 WinRunner 中通过形象规划工具函数发生器能很快的增加测试脚本的功能。在 TSL 中涉及 TSL 和 TSL 参考指导，函数发生器的解释说明中所有的函数都有介绍。

函数发生器能够让测试人员使用两种方法增加 TSL 函数：

- 指向 GUI 对象和让 WinRunner“建议”一个适当的函数。把函数插入到测试脚本中。
- 能从下拉表中挑选函数。函数显示受种类和字母排序的区别。

为了更进一步提高测试脚本的逻辑性。简单的类型规划作为条件声明，循环和算法，操作员可以通过他立即到测试窗口中。

下面是创造一个测试的脚本：

打开命令、打开传真对话框命令、检查结果相等票的数量和次序增加入场卷价格，录制一个基本的测试脚本。

1.录制打开飞行预约程序和打开传真命令框

如果 WinRunner 没有打开，选择 Programs>WinRunner>WinRunner,如果欢迎窗口打开了，就点击 New Test 按钮。否则，选择 File>New. 一个新的测试打开了。

2.开始飞行预约程序并注册

选择 Programs>WinRunner>Sample Applications>Filght 1A

3.开始录制上下文敏感模式

选择 Create>Record-Context Sensitive 或点击 Record 按钮

4.打开命令#4

在飞行预约程序中，选择 File>Open Order.打开命令对话框，挑选 Order No.检查对话框和打字“4”在邻近的对话框中，点击 OK 来打开命令。

5.打开传真对话框

选择 File>Fax Order

6.点击取消按钮关闭对话框

7.停止录制

8.保存测试

9.如果工作在全局 GUI 模式中，就需要保存新的 GUI Map 对象文件。

选择 Tools>GUI Map Editor.选择 View>GUI Files.选择 File>Save.点击 yes 或 no 和增加新的窗口到的 GUI 文件中，选择 File>Exit 关闭 GUI 文件编辑。

10．2 使用函数发生器插入函数

测试人员可以通过函数发生器为脚本定义函数或则通过其选择相应的函数。这里准备使用函数发生器为上面的测试脚本增加函数

1.在 button_press("Cancel");的上面插入一个空白的行声明指针开始的行

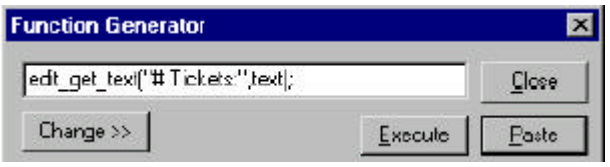
2.打开“传真命令”对话框

选择 File>Fax Order 在飞行预约窗口

3.询问这个#票的领域

选择 Create> insert Function>For object/Window 或点击 insert Function for Object/Window 按钮在工具栏。使用手指指示器点击#票的领域。

这个函数发生器打开建议 edit_get_text 函数。如图所示



如图 10-1

这个函数在＃票的领域中读取文本和赋值给变量。这个默认值的变量的名字是 Text.改变变量的名字 text 到 ticket 被键入到：

```
edit_get_text("#Tickets:",tickets);
```

点击 Paste 增加函数的测试脚本。

4.入场卷价格的疑问

选择 Create>insert Function>For Object/Window 或点击 insert Function for Object/Window 按钮。使用手指指示器点击票的价格。

这个函数发生器打开和建议 edit_get_text 函数。改变 text 变量的名字到 price:

```
edit_get_text("Tlicket  Price:",price);
```

点击 Paste 增加函数到脚本中。

5.询问总计的 Field

选择 Create>insert Function>For Object/Window 或点击 insert Function For Object/Window 按钮，使用手指指示器点击总计 Field

这个函数发生器打开和建议 edit_get_text 函数。改变名字 text 变量到 total：

```
edit_get_text("Total:",total);
```

点击 Paste 给增加函数到测试脚本

6.关闭传真命令对话框

点击 Cancel 关闭对话框在飞行预约程序。

7.保存测试

选择 File>Save 或点击保存按钮。

8.如果工作在全局 GUI 模式中，保存新的 GUI Map 对象。

选择 Tools>GUI Map Editor.选择 View>GUI Files.选择 File>Save.点击 Yes 增加新的对象或 GUI map 窗口。打开一个 WinRunner 消息框。点击 OK。选择 File>Exit 关闭 GUI 图形编辑。

10.2.1 增加测试脚本的逻辑

在这个练习要编辑逻辑脚本。在测试脚本中使用一组 if/else 语句：

- 检查每张票的单价乘以预定的票数得到的结果是否和总数相同；
- 在报表中显示总数是否正确

1.将鼠标指针移到上述测试脚本中最后一条 edit_get_text 语句后；

2.把下列语句正确地插入到测试脚本中。

```
if(tickets*price = = total)
```

```
                tl_step("total",0,"Total is correct.");
        else
                tl_step("total",1,"Total is incorrect.");
```

这些语句的意思是：“如果 tickets 和 price 相乘的值等于 total，报告总数是正确的，否则报告总结是错误的。

附录 A（自定义键）

命令	默认自定义键	功能
录制	F2	**开始录制（包括** Context Sensitive **和** Analog **）**
检查 GUI(Single)	Alt Right+F12	**检查单独的** GUI **对象的属性**
检查 GUI（Object/Window）	Ctrl Right+F12	**检查对象或窗口的** GUI **对象检查点**
检查 GUI(Multiple Objects)	F12	**打开创建** GUI **检查点对话框**
检查 Bitmap(Object/Window)	Ctrl Left+F12	**捕获一个对象或窗口位图**
检查 Bitmap（Screen Area）	Alt Left +F12	**捕获一个范围的位图**
检查 DataBase(Default)	Ctrl Right +F9	**在全部的数据内容中创建一个检查**
检查 DataBase(Custom)	Alt Right +F9	**在数据库中检查纵号和指定行的信息**
同步 Prorerty(Object/Window)	Ctrl Right F10	**教** WinRunner **等待一个对象或窗口中期待的属性值出现**
同步 Bitmap（Object/Window）	Ctrl Left +F11	**教** WinRunner **等待指定的对象或窗口位图出现**
同步 Bitmap(Screen Area)	Alt Left +F11	**教** WinRunner **等待指定范围的位图显示**
Get Text From Object/Window	F11	**捕获一个对象或窗口的文本**
Get Text From Window Area	Alt Right+F11	**捕获指定范围的文本并增加** obj_get_text **声明在测试脚本中**
Get Text From Screen Area	Ctrl Right +F11	**捕获指定范围的文本并增加** get_text **声明到测试脚本中**
Insert Function For Object/Window	F8	**在** GUI **对象中插入一个** TSL **函数**
Insert Function From Function Generator	F7	**打开函数发生器对话框**
Run From Top	Ctrl Left +F5	**从开始运行测试**
Run From Arrow	Ctrl Left +F7	**从指示箭头行开始运行测试**
Step	F6	**单步运行**
Step Into	Ctrl Left +F8	**调用测试或函数显示在** WinRunner**，但是函数没有被执行**

Step to Cursor	Ctrl Left +F9	**从指针单步运行**
暂停	Puse	**暂停**
停止	Ctrl Left +F3	**停止运行测试**
Move Locator	Alt Left +F6	Records a move_ locator_ abs statement with the current position (in pixels) of the screen pointer.

附录 B（API 函数）

Setvar(),getvar()
attached_ text_ area (Miscellaneous\ Attached Text \ Preferred search area)
attached_ text_ search_ radius (Miscellaneous \ Attached Text \ Search radius)
batch (Run \ Run in batch mode，**默认值为** 0)
beep (Run \ Beep when checking a window， **默认值为** on)
cs_ fail (Run \ fails a test when Context Sensitive errors occur，**默认值为** 0)
cs_ run_ delay (Run \ Advanced \Delay between ezecution of CS statements，**默认值为** 0)
curr_ dir (Current folder，**没有默认值**)
delay_ msec(Run \ Delay for window synchronization，**默认值为** 1000)
drop_ sync_ timeout (Run \ Advanced \ Drop synchronization timeout if failed，**默认值为** selected)
enum_ descendent_ toplevel (Record \ Advanced \ Consider child windows，**默认值位** on)
exp (Expected results folder，**没有默认值**)
fontgrp(Text Recognition \ Font group，**默认值位** Stand)
item_ number_ seq (Miscellaneous \ String indicating that what follows is a number，**默认值为**#)
key_ editing (Record \ Advanced \ Generate concise, more readable type statements，**默认值为** on)
line_ no
list_ item_ separator
listview_ item_ separator
min_ diff
mismatch_ break
rec_ item_ name
rec_ owner_ drawn
result
runmode
searchpath
shared_ checklist_ dir
single_ prop_ check_ fail
silent_ mode
speed
sync_ fail_ beep
synchronization_ timeout
td_ log_ dirname

td_ connection
td_ cycle_ name
td_ database_ name
td_ server_ name
td_ user_ name
tempdir
testname
timeout_ msec
treeview_ path_ separator

例如：

```
t = getvar (" timeout_ msec");
d = getvar (" delay_ msec");
setvar (" timeout_ msec", 30000);
setvar (" delay_ msec", 3000);
win_ check_ bitmap (" calculator", Img1, 2, 261, 269, 93,42);
setvar (" timeout_ msec", t);
setvar (" delay_ msec", d);
```

定制 WinRunner Function Generator
增加 Function Generator **的种类：**generator_ add_ category (category_ name);

附录 C（部分属性设置）

Text Recognition Tab
Put Recognized Text in Remark:该选项是设置在设置检查点的时候是否添加相应的注释
Timeout for Text Recognition:该选项是设置识别文本的最大间隔时间（利用 text checkpoint）
Use Image Text Recognition Mechanism:该选项是是否选择文本肖像识别机制
Remove Spaces from Recognized Text:该选项是删除或传输多个加载和跟踪表格空间（必须重新启动改变 WinRunner 设置的结果）
Font Group:这个能够使肖像文本承认（上面的描述），这个选项设置实激活字体群。

附录 D(在 WinRunner 中识别 Dephi 控件)

 # 制作环境：

 # WinRunner 7.0 Build 7211

 # Delphi 6.0

 # 方法发现： 袁海松

```
# 制 作： 杨欣源

# 使用方法：

# Step 1 拷贝本文档中所有内容

# Step 2 打开 WinRunner\lib\vbinit 中的 script 文件

# Step 3 将内容粘贴在段落“

# VisualBasic 4.0 Standart controls”之前

# Step 4 重新运行 WinRunner

# Step 5 启动时装载 VB Addin

# TBitBtn

set_class_map("tbitbtn", "push_button"); set_record_attr("tbitbtn", "class label
x y", "MSW_id", "location");

# TButton

set_class_map("tbutton", "push_button"); set_record_attr("tbutton", "class label
x y", "MSW_id", "location");

# TCheckBox

set_class_map("tcheckbox", "check_button"); set_record_attr("tcheckbox", "class
label x y", "MSW_id", "location");

# TComboBox

set_class_map("TComboBox", "combobox");db2inst1 set_record_attr("TComboBox",
"class attached_text x y", "MSW_id", "location");

# TDateTimePicker

set_class_map("tdatetimepicker", "sysdatetimepick32");
set_record_attr("tdatetimepicker", "class attached_text x y", "MSW_id",
"location");

# TEdit

set_class_map("tedit", "edit"); set_record_attr("tedit", "class attached_text x y",
"MSW_id", "location");
```

```
# TGroupButton

set_class_map("tgroupbutton", "radio_button"); set_record_attr("tgroupbutton",
"class label x y", "MSW_id", "location");

# TListBox

set_class_map("TListBox", "listbox"); set_record_attr("TListBox", "class
attached_text x y", "MSW_id", "location");

# TListView

set_class_map("TListView", "syslistview32"); set_record_attr("TListView", "class
attached_text x y abs_x", "MSW_id", "location");

# TMaskEdit

set_class_map("TMaskEdit", "edit"); set_record_attr("TMaskEdit", "class
attached_text x y", "MSW_id", "location");

# TMemo

set_class_map("tmemo", "edit"); set_record_attr("tmemo", "class attached_text xy",
"MSW_id", "location");

# TMonthCalendar

set_class_map("TMonthCalendar", "sysmonthcal32");
set_record_attr("TMonthCalendar", "class attached_text x y", "MSW_id",
"location");

# TRadioButton

set_class_map("TRadioButton", "radio_button"); set_record_attr("TRadioButton",
"class label x y", "MSW_id", "location");

# TRickEdit

set_class_map("TRichEdit", "edit"); set_record_attr("TRichEdit", "class
attached_text x y", "MSW_id", "location");

# TPageControl

set_class_map("tpagecontrol", "systabcontrol32"); set_record_attr("tpagecontrol",
"class attached_text x y", "MSW_id", "location");
```

```
# TStaticText

set_class_map("TStaticText", "object"); set_record_attr("TStaticText", "class
regexp_MSW_class label", "attached_text MSW_id MSW_class", "location");

# TTreeView

set_class_map("ttreeview", "systreeview32"); set_record_attr("ttreeview", "class
attached_text", "MSW_id", "location");
```